

Zur Theorie des Look-ahead-Addierers

Dissertation
zur Erlangung des Grades eines
Doktor-Ingenieurs
der Abteilung für Elektrotechnik
an der Ruhr-Universität Bochum

von
Karl Zeiske
Berlin

Bochum 1972

Zur Theorie des Look-ahead-Addierers

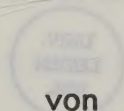
Dissertation

zur Erlangung des Grades eines

Doktor-Ingenieurs

der Abteilung für Elektrotechnik

an der Ruhr-Universität Bochum



von

Karl Zeiske

Berlin

Bochum 1972



Dissertation eingereicht am: 13.4.1972

Tag der mündlichen Prüfung: 14.11.1972

Referent: Prof. Dr. phil. nat. Schneider

Korreferent: Prof. Dr. Ing. Leilich

Vorwort

Diese Arbeit entstand aus Literaturstudien und Überlegungen, die zur Vorbereitung der im Wintersemester 1971/72 gehaltenen Vorlesung Informationstechnik III (Datenverarbeitung) notwendig waren.

Ich möchte Herrn Prof. Dr. phil. nat. Schneider für die Förderung dieser Arbeit sehr danken und diesen Dank darauf ausweiten, daß er mir während der langjährigen Zusammenarbeit immer Gelegenheit gegeben hat, diesem meinem selbstgewählten Spezialgebiet neben meiner Tätigkeit als Oberingenieur am Lehrstuhl für Regelungstechnik nachzugehen.

Herrn Dipl.-Ing. Thiele möchte ich für die Diskussionsmöglichkeiten bei der Vorlesungsvorbereitung, in die auch die Diskussionen über die Addierer eingeschlossen sind, danken.

Herrn Günther Leder möchte ich für die schnelle und sorgfältige Herstellung der Zeichnungen ebenfalls in diese Danksagung einschließen.

Inhaltsverzeichnis

Bezeichnungen und Bezüge	5
Einleitung	7
Der Parallel-Addierer	8
Der Look-ahead-Addierer	11
Maximales FAN-IN = $n+1$	15
Addierer ohne Hilfsfunktionen	17
Hilfsfunktionen 1. Art	20
Hilfsfunktionen 2. Art	27
Hilfsfunktionenpaare p-ter Art	30
Gesamtrechnzeit als Funktion von m und n	34
Reduzierung der Elektronik	37
Die Elektronik-Tabellen	39
Fixierung nicht voller Hilfsfunktionen	41
Berechnung des Elektronikaufwands	45
Die Addierer-Spitze	47
Die 1. Stufe	48
Zusätzliche Hilfsfunktionenpaare	50
Ineinanderschieben von Stufen	52
Der Carry Look-ahead-Addierer nach MacSorley	55
Der Look-ahead-Addierer nach Flores	61
Der Addierer nach Weinberger und Smith	65
Methodenvergleich an einem 25-Bit-Addierer	68
Berücksichtigung von FAN-OUT-Zahlen	74
Zusammenfassung	77
Anlagen	79
Literatur	81

Bezeichnungen und Bezüge

A_i	i-tes Bit des Akkumulators
D_i	i-tes Bit des Daten-Registers
C_i	i-tes Carry (Übertrag)
F_i	i-te Fortpflanzungsfunktion
G_i	i-te Generate-Funktion
$C(i)$	Carry-Funktion, die aus 1 Carry und i-1 Funktionenpaaren gebildet wird
$FG(i)$	Hilfsfunktionenpaar, das aus i Funktionenpaaren gebildet wird
$C(j)_i$	i-tes Carry, das aus 1 Carry und j-1 Funktionenpaaren gebildet wird
m	Länge des Addierers in Bit
n	Länge der 1. Stufe oder einer einfachen Carry-Look-ahead-Gruppe
r	obere Indizes bei Hilfsfunktionen
s	untere Indizes bei Hilfsfunktionen
S_i	i-tes Summen-Bit
T_m	Gesamtadditionszeit des Addierers mit m Bit
σ	Durchlaufzeit durch ein Logisches Element
τ	Zeit zwischen der Bildung zweier nacheinander gebildeter Carry

	Seite		Seite		Seite
Abb. 1	14	Abb. 5	35	Anlage 1	79
Abb. 2	19	Abb. 6	53	Anlage 2	80
Abb. 3	25	Abb. 7	54	Tabelle I	8
Abb. 4	29	Abb. 8	60	Tabelle II	40

Bezug	Seite	Bezug	Seite
(1)	8	(23)	42
(1')	9	(24), (25)	43
(2), (3)	9	(26)	46
(3'), (3'')	15	(27)	47
(3''')	61	(28), (29)	48
(4') bis (4''')	14	(30), (31)	50
(5')	14	(32)	54
(5'')	13, 25	(33), (33')	65
(6'), (6'')	13, 17, 25, 31	(34), (35)	66
(7), (8)	15	(36), (37)	68
(9)	17, 25, 71	(38)	69
(10)	20	(39)	69, 71
(11)	22	(40)	69
(12)	23	(41)	69, 71
(13), (14)	25	(42)	69
(16), (17)	39	(43)	69, 71
(18), (19)	40	(44), (45), (46)	70
(20), (21), (22)	41	(47)	71

Einleitung

1956 haben Weinberger und Smith in [1] gezeigt, daß es möglich ist, die Additionszeit eines Parallel-Addierers dadurch auf wenige Prozente zu reduzieren, daß die Überläufe (Carrys) einer Anzahl von Bits nicht nacheinander, sondern in Gruppen gleichzeitig (simultan) berechnet werden. Den Autoren lagen spezielle Elektronik-Elemente (OR-AND-OR-Gatter mit speziellen maximalen FAN-IN-Zahlen) vor, die beim Aufbau des Addierers zu bestimmten nicht verallgemeinerungsfähigen Gruppierungen führten.

1961 gab MacSorley in [2] eine Version eines solchen Addierers an, aufgebaut aus selbständigen OR- und AND-Gattern. Spätestens von hier an wird der Name Look-Ahead-Addierer verwendet.

1963 faßt Flores in [3] das vorliegende Wissen zusammen, berücksichtigt aber nicht mehr maximale FAN-IN-Zahlen, wie sie bei MacSorley noch fast wie von selbst eingehalten erscheinen. Flores übernimmt auch Teile des Aufbaues nach MacSorley, Richtlinien aber, aus denen zu ersehen ist, wann Hilfsfunktionen nötig sind, und bis zu welchem Grade, werden nicht angegeben.

In der vorliegenden Arbeit soll die Gliederung eines Look-Ahead-Addierers als Funktion einer maximalen FAN-IN-Zahl dargestellt werden. Es werden sich daraus Regeln für den Einsatz von Hilfsfunktionen ableiten lassen, ebenfalls Regeln für das Einsparen von Elektronik.

Anschließend wird über die Arbeiten von MacSorley [2], Flores [3] und Weinberger und Smith [1] berichtet, und dann an einem 25-Bit-Addierer die Unterschiede der Methoden demonstriert. Den Abschluß bilden einige Überlegungen zum Einfluß der FAN-OUT-Zahl.

Der Parallel-Addierer

Der Inhalt des Akkumulators eines Digitalrechners lässt sich in der Form

$$A = A_m \cdot 2^{m-1} + A_{m-1} \cdot 2^{m-2} + \dots + A_1 \cdot 2^0$$

darstellen, wobei die A_i die Werte 0 (Null) und 1 (Eins) annehmen können. Der Inhalt des Daten-Registers

$$D = D_m \cdot 2^{m-1} + D_{m-1} \cdot 2^{m-2} + \dots + D_1 \cdot 2^0$$

A_i	D_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabelle I

mit $D_i = 0$ oder 1 soll zum Inhalt des Akkumulators addiert werden. Werden mit S_i die zu errechnenden Werte der Summen-Bit bezeichnet, so müssen beim Parallel-Addierer m logische Gleichungen für die Bestimmung der S_i , $i=1, \dots, m$ und die gleiche Zahl von logischen Gleichungen für die Bestimmung der Überträge (Carry) C_i aufgestellt werden. Für die S_i gilt entsprechend der Tab. 1

$$S_i = \overline{A_i} \overline{D_i} C_{i-1} + \overline{A_i} D_i \overline{C_{i-1}} + A_i \overline{D_i} \overline{C_{i-1}} + A_i D_i C_{i-1} \quad (1)$$

Für die Überträge gilt

$$C_i = \overline{A_i} D_i C_{i-1} + A_i \overline{D_i} C_{i-1} + A_i D_i \overline{C_{i-1}} + A_i D_i C_{i-1} \quad .$$

Die letzte Gleichung lässt sich vereinfachen zu

$C_1 = A_1 D_1 + (A_1 + D_1) C_{1-1}$, woraus sich mit Einführung der für Look-Ahead-Addierer besonders geeigneten Funktionen $G_1 = A_1 B_1$ und $F_1 = A_1 + B_1$ sofort

$$C_1 = G_1 + F_1 C_{1-1} \quad (2)$$

ergibt.

Dabei steht G für Generate ein Carry, wenn A_i und D_i zugleich L sind, und F für Fortpflanze ein Carry, falls mindestens eines der A_i oder D_i gleich L ist und außerdem noch das vorhergehende Carry ansteht.

Die Gleichung (1) kann mit den Funktionen G_i und F_i die Form

$$S_i = \overline{F_i} C_{i-1} + \overline{G_i} F_i \overline{C_{i-1}} + G_i C_{i-1} \quad (1')$$

annehmen.

Die Gleichungen (1') sind im oberen Drittel der Abb. 1 durch logische Elemente dargestellt, wobei die Negationen durch kleine schwarze Punkte an den Eingängen symbolisiert sind. Diese Darstellung ist bewußt an die Darstellung in [2] angelehnt.

Die Gl. (2) sind in der Mitte der Abb. 1 dargestellt und um einen Addierer zu erhalten, muß man sich die gleichnamigen Größen des oberen und des mittleren Teils durch senkrechte Striche verbunden denken.

Aus der Abb. 1, mittlerer Teil, sowie aus der Folge der Gleichungen

$$\begin{aligned} C_1 &= G_1 + F_1 \cdot C_0 \\ C_2 &= G_2 + F_2 \cdot C_1 \\ C_3 &= G_3 + F_3 \cdot C_2 \end{aligned} \quad (3)$$

ist zu ersehen, daß jedes C_i immer erst gebildet werden kann, wenn

vorher C_{i-1} gebildet wurde. Zur Bildung von C_i muß C_{i-1} (s. Abb. 1) ein AND- und ein OR-Glied durchlaufen. Nimmt man als Durchlaufzeit durch jedes dieser Glieder die Zeit σ an, so wird jedes C_i 2σ später gebildet als das vorhergehende C_{i-1} , z.B. C_2 nach 5σ , da C_1 nach 3σ ansteht. Eine Ausnahme macht C_1 , da hier die Zeit 3σ durch F_1 und G_1 bestimmt wird, die selbst erst nach der Zeit 1σ gebildet sind, und dann noch durch 2 weitere Glieder zur Bildung von C_1 laufen müssen. In Abb. 1 sind die Bildungszeiten an den Ausgängen der Carry C_1, C_2, C_3 eingetragen. Allgemein kann gesagt werden, daß die Bildungszeit für C_i gleich $(2i+1)\sigma$ ist.

Beachtet man, daß zur Bildung des S_i der Wert von C_{i-1} auch negiert benötigt wird und setzt man für die Ausführung dieser Negation ebenfalls die Zeit σ an, so erfolgt die Bildung der S_i jeweils 3σ später als die Bildung des zugehörigen C_{i-1} , allgemeingesagt wird S_i nach der Zeit $(2i+2)\sigma$ gebildet. Unter den Ausgängen S_i ist in Abb. 1 die dazugehörige Bildungszeit eingetragen.

Ein Addierer mit m Bit benötigt also für die Ausführung einer Addition die Zeit $T_m = (2m+2)\sigma$. Diese Zeit T_m ist ein Maß, an dem alle Beschleunigungen der Additionszeit gemessen werden können. Ein weiteres Zeitmaß, daß außerdem von der Bitzahl des Addierers und auch weitgehend von den speziellen Eigenschaften der verwendeten Elektronik unabhängig ist, ist die Zeitspanne, die bei diesem Addierer zwischen der Bildung zweier aufeinanderfolgender Carrys liegt. Sie soll τ genannt werden und in diesem Fall bei Verwendung von OR- und AND-Gliedern mit je einer Durchgangszeit σ , ist $\tau = 2\sigma$.

Der Look-Ahead-Addierer

Ersetzt man in (3) auf der rechten Seite successiv die Werte C_1 und C_2 durch die C_1 und C_2 definierenden Gleichungen, so entsteht

$$\begin{aligned} C_1 &= G_1 + F_1 C_0 \\ C_2 &= G_2 + F_2 (G_1 + F_1 C_0) = G_2 + F_2 G_1 + F_2 F_1 C_0 \\ C_3 &= G_3 + F_3 (G_2 + F_2 G_1 + F_2 F_1 C_0) = G_3 + F_3 G_2 + F_3 F_2 G_1 + F_3 F_2 F_1 C_0 \end{aligned} \quad (3')$$

Dies läßt sich allgemein zu folgender Formel für C_i erweitern:

$$C_i = G_i + F_i G_{i-1} + F_i F_{i-1} G_{i-2} + \dots + F_i F_{i-1} \dots F_2 F_1 C_0 \quad (3'')$$

Betrachtet man in Abb. 1 die unterste Anordnung, so findet man das Logik-Bild für die Gl. (3'). Da angenommen wird, daß alle F_i und G_i nach der Zeit σ zur Verfügung stehen, folgt für alle C_i , daß sie nach der Zeit 3σ zur Verfügung stehen. Dementsprechend sind die dazugehörigen S_i nach der Zeit 6σ gebildet. Eine Ausnahme macht nur S_1 mit 4σ . Man kann sich den Addierer dadurch gebildet denken, daß die gleichbenannten Größen des oberen Teils und des untersten Teils der Abb. 1 verbunden gedacht werden.

Es ist in Abb. 1 zu sehen, daß mit steigender Indexzahl i die Zahl der logischen Elemente für die Bildung von C_i zunimmt. Das gleiche folgt aus (3''). Gäbe es keine Beschränkungen in Form von maximalen FAN-IN-Zahlen und in dem Maß an Elektronik, die verwendet werden soll, so könnten alle Parallel-Addierer das Ergebnis in der Zeit 3τ bei beliebiger Bitzahl m liefern. Gegenwärtig ist aber die maximale FAN-IN-Zahl die wesentliche Beschränkung, die zu längeren Additionszeiten führt. Selbstverständlich existiert auch eine maximale FAN-OUT-Zahl, in der Regel kann aber diese Zahl durch parallele Logik-Elemente überspielt werden. In einem speziellen Beispiel soll darauf noch eingegangen werden.

Obwohl ein Addierer mit der Arbeitszeit 3τ in der Regel nicht realisiert werden kann, so soll der erforderliche Aufwand an Elektronik berechnet werden, da hiermit eine obere Grenze für den elektro-

nischen Aufwand beim Look-Ahead-Addierer zu bekommen ist.

Der Abb. 1, unteres Drittel, ist zu entnehmen, daß zur Bildung von

$$m \text{ Werten } C_1 \text{ } m \text{ OR-Glieder} \quad (4')$$

benötigt werden. Weiter werden

$$\sum_{i=1}^m i = (m+1)m/2 \text{ AND-Glieder} \quad (4'')$$

benötigt. Nimmt man als weiteres Maß für die elektronische Komplexität die Anzahl der Verbindungsdrähte, oder die Zahl der Eingangsdioden, die der ersten Zahl gleich ist, so erhält man für die Zahl der Eingänge an den OR-Gliedern

$$\sum_{i=2}^{m+1} i = (m+3)m/2 \quad (4''')$$

und für die Zahl der Eingänge an den AND-Gliedern, die zur Bildung eines C_1 notwendig sind, $\sum_{j=2}^{i+1} j$ und daraus für die Zahl der Eingänge aller AND-Glieder

$$\begin{aligned} \sum_{i=1}^m \sum_{j=2}^{i+1} j &= \sum_{i=1}^m ((i+3)i/2) = \frac{1}{2} \sum_{i=1}^m i^2 + \frac{3}{2} \sum_{i=1}^m i = \\ &= \frac{1}{2} \frac{1}{6} (m+1)m(2m+1) + \frac{3}{2} (m+1)m/2 = \frac{1}{2} (m+1)m \left(\frac{1}{6}(2m+1) + \frac{3}{2} \right) = \\ &= \frac{1}{6} (m+1)m(m+5) \quad (4''') \end{aligned}$$

Für $m = 25$ werden daher 25 OR-Glieder, 325 AND-Glieder und 350 Dioden an den OR-Gliedern und 3250 Dioden an den AND-Gliedern benötigt. Faßt man die beiden letzten Zahlen zusammen, so läßt sich das Tripel

$$25, \quad 325, \quad 3600 \quad 3\tau \quad (5')$$

leicht verstehen. Die erste Zahl gibt die Anzahl der OR-Glieder, die zweite Zahl die Anzahl der AND-Glieder, und die dritte Zahl ist ein Maß für die Komplexität des Addiereraufbaues. Die theoretische Gesamtrechnzeit ist dann noch hinzugefügt. Bildet man die entsprechenden Zahlen für die langsamste Version des Parallel-addierers, so erhält man , entsprechend dem mittleren Teil der Abb. 1, für $m = 25$

$$25, \quad 25, \quad 100 \qquad 26\tau \quad . \qquad (5'')$$

Diese Zahlen stehen eigentlich isoliert und sie sollten um den in jedem Fall gleichen Aufwand für die Bildung der F_i , G_i und S_i vermehrt werden.

$$50, \quad 100, \quad 350 \qquad (5''')$$

sind die entsprechenden Zahlen, da in Abb. 1, oberes Drittel, für jedes 1 2 OR-, 4 AND-Glieder und 14 Dioden benötigt werden.

Bildet man entsprechende neue Tripel, so erhält man für das eine theoretische Extrem (größter Aufwand, kürzeste Rechenzeit) aus (5') und (5''')

$$75, \quad 425, \quad 3950 \qquad 3\tau \qquad (6')$$

und für das andere Extrem (kleinster Aufwand, längste Rechenzeit) aus (5'') und (5''')

$$75, \quad 125, \quad 450 \qquad 26\tau \quad . \qquad (6'')$$

Alle weiteren Ergebnisse über Rechengeschwindigkeiten und Elektronikaufwand müssen in den Grenzen zwischen (6') und (6'') liegen.

Die eigentliche Beschränkung bei der Realisierung des 3τ Addierers liegt, wie schon erwähnt, in der Beschränktheit der maximalen FAN-IN-Zahl der elektronischen Bauelemente. Die Untersuchung,

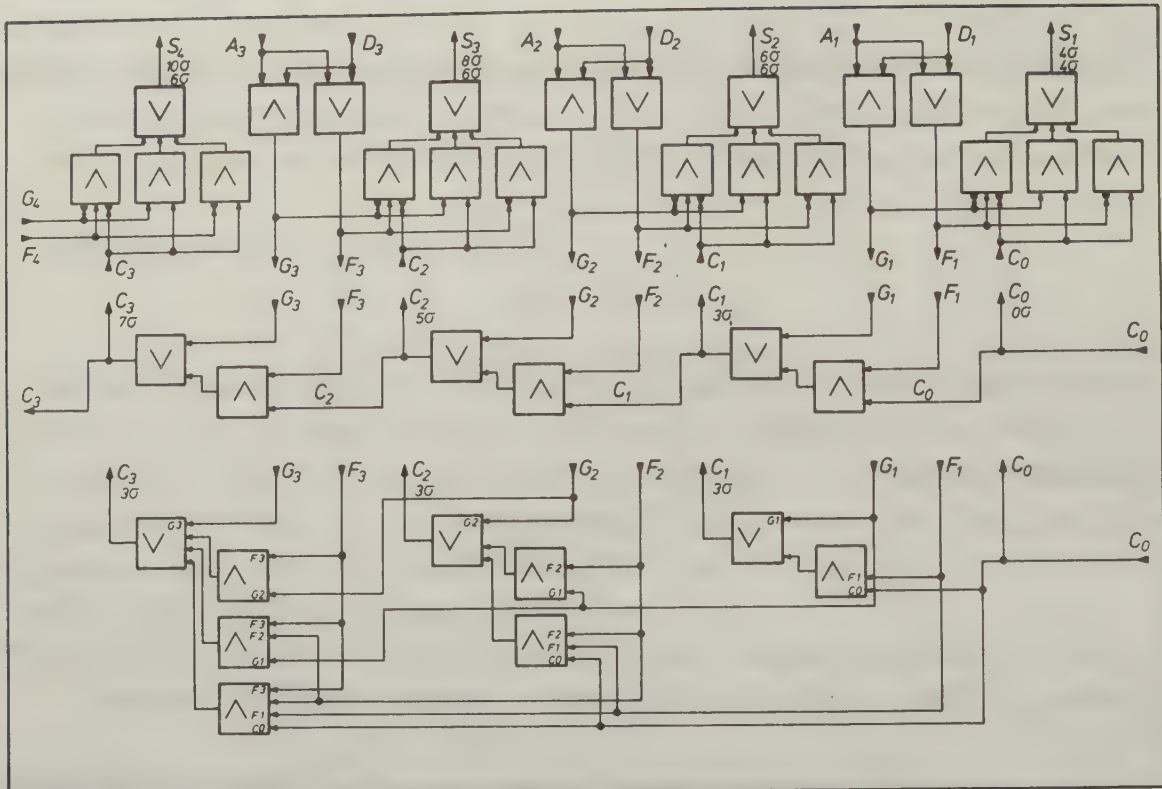


Abb. 1 Prinzip-Schaltung für F_1 , G_1 , C_1 und S_1

welche Rechengeschwindigkeit mit einer definierten maximalen FAN-IN-Zahl möglich ist, bildet den Inhalt der nächsten Abschnitte.

Maximal FAN IN = N + 1

Im Weiteren sei angenommen, daß die maximale FAN-IN-Zahl sowohl der OR-Glieder als auch der AND-Glieder den gemeinsamen Wert $n + 1$ besitzt. Es folgt dann aus (3''), daß unter dieser Voraussetzung gerade noch

$$C_n = G_n + F_n G_{n-1} + \dots + F_n F_{n-1} \dots F_2 F_1 C_0 \quad (7)$$

gebildet werden kann. Der letzte Term besteht aus $F_1, F_2, \dots, F_n$ und C_0 , das sind gerade $n + 1$ Eingänge an diesem AND-Gatter, die jeweils letzten Faktoren der Terme werden aber von $G_n, G_{n-1}, \dots, G_1, C_0$ gebildet, d.h. es existieren genau $n + 1$ Terme und damit ebensoviel Eingänge an dem OR-Glied, das C_n bildet. Dagegen würde die analoge Bildung von C_{n+1} die maximale FAN-IN-Zahl überschreiten.

Im langsamsten Parallel-Addierer wurde C_n zur Bildung von C_{n+1} , dieser zur Bildung von C_{n+2} , usw. benutzt. Es liegt nahe, C_{n+2} wieder gleichzeitig mit C_{n+1} aus C_n bilden zu lassen, die Gleichungen für die nächsten n Werte C_i für $i > n$ lauten dann

$$\begin{aligned} C_{n+1} &= G_{n+1} + F_{n+1} C_n \\ C_{n+2} &= G_{n+2} + F_{n+2} G_{n+1} + F_{n+2} F_{n+1} C_n \\ &\vdots \\ &\vdots \\ &\vdots \\ C_{2n} &= G_{2n} + F_{2n} G_{2n-1} + \dots + F_{2n} F_{2n-1} \dots F_{n+1} C_n \end{aligned} \quad (8)$$

In der Gleichung für C_{2n} ist wieder die maximale FAN-IN-Zahl erreicht, da die letzte der Gln. (8) aus Gl. (7) dadurch gebildet werden kann, daß alle Indizes um n erhöht werden. Also gilt die Überlegung für (7) auch hier.

Alle Gln. (8) können die C_i erst 2σ nach der Bildung von C_n bilden, d.h. die Gesamtadditionszeit für Addierer mit der Bitzahl m und $n < m \leq 2n$ würde 4τ zur Bildung des Ergebnisses benötigen.

Es soll noch festgehalten werden, daß sowohl C_1 wie C_{n+1} aus je einem Paar F, G -Funktionen und einem abschließendem C gebildet werden. Für C_k und C_{n+k} werden neben dem abschließendem C genau k Paare von F, G -Funktionen verwendet, so daß für $k = n$ genau n Paare von F, G -Funktionen verwendet werden. Man kann daher die Eigenschaft, die maximale FAN-IN-Zahl sei $n + 1$, so umformulieren:

Zur Bildung eines Carrys C_1 dürfen neben einem weiteren notwendigen C_t höchstens n Paare von F, G -Funktionen verwendet werden.

Daß die Indizes dieser Funktionspaare lückenlos in dem Bereich von $t + 1$ bis i liegen müssen, ist leicht aus (8) oder (3') und (7) zu entnehmen.

Addierer ohne Hilfsfunktionen

Der eben angegebene Übergang von C_n zu C_{2n} läßt sich fortsetzen und man kann z.B. die C_{2n+1} bis C_{3n} als Funktionen von C_{2n} bilden usw. und bei jeder neuen Gruppe von n Werten C_1 wird die Gesamtrechnenzeit um τ vergrößert. Allgemein kann gesagt werden, daß ein Addierer, der aus m Bit besteht, in k Gruppen geteilt werden kann, wobei $k = m/n$ ist, falls n ein Teiler von m ist, oder sonst $k = 1 + \text{ENTIER}(m/n)$ ist. Die Gesamtrechnenzeit ist gleich $(k+2) \cdot \tau$ oder $2\tau + T_m/n$. Für einen 25-Bit-Addierer und für $n = 5$ ergeben sich so 5 Gruppen oder Stufen und eine Rechenzeit von 7τ gegenüber den 26τ des einfachen Parallel-Addierers. Berechnet man den Aufwand an Elektronik nach den Formeln (4') bis (4''') für $m = 5$ für eine Stufe und multipliziert dann mit 5 für den Aufwand aller Stufen, so erhält man zuerst

5, 15, 70	für eine Stufe	und daraus
25, 75, 350	für die Carry-Bildung aller Stufen.	

Wird diese Zahl mit (5''') vergrößert, um den Gesamtaufwand für den Addierer zu übersehen, und vergleicht man mit (6') und (6'') so entsteht

75, 425, 3950	3τ	nicht erreichbar	(6')
75, 175, 700	7τ	5 x 5 Gruppen	(9)
75, 125, 450	26τ	langsamster Addierer	(6'')

Durch einen Aufwand von 40% mehr AND-Gattern und knapp 80% mehr Dioden und Drähten läßt sich die Rechenzeit bei diesem 25-Bit-Addierer von 26τ auf 7τ herabdrücken, wobei immer ein maximales FAN-IN von 6 zugrunde lag. Wäre diese Zahl 7, so käme keine schnellere Rechenzeit heraus, da $25/6 = 4,16$ ebenfalls zu $k = 5$ führt. Erst mit maximalem FAN-IN von 8 und Gruppengröße 7 könnte die Rechenzeit wegen $k = 4$ auf 6τ herabgedrückt werden.

Eine Übersicht über einen Addierer, der 25 Bit umfaßt und der aus 5 Gruppen zu je 5 Bit für die Carrybildung besteht, zeigt Abb.2.

Die Zeitachse ist von oben nach unten laufend angenommen und die Vielfachen von σ sind eingetragen. Die einzelnen Elemente für die Bildung der Carrys sind entsprechend der Zeitachse zu den Zeitpunkten eingetragen worden, an denen die Bildung der jeweiligen Carrys abgeschlossen ist. Die Bitnummern i sind über den Kästen, in denen die Funktionen F_i und G_i gebildet werden, eingetragen. Für die beiden Funktionen F_i und G_i ist immer nur eine Verbindungslinie gezogen, da diese Funktionen nur paarweise benutzt werden. Ein Carry-Kasten C , in dem von oben kommend 3 Verbindungslinien enden, besitzt also eine logische Gleichung, die 2 Paare von F, G -Funktionen und ein Carry verwendet, z.B.

$C_i = G_i + F_i G_{i-1} + F_i F_{i-1} C_{i-2}$. Da in einem Carry maximal n Paare F, G -Funktionen zur Einhaltung der maximalen FAN-IN-Zahl verarbeitet werden können, können in einen Kasten also nur $n + 1$ Verbindungslinien von oben einmünden.

Die S_i sind in Abb. 2 nicht entsprechend dem Zeitpunkt eingetragen worden, an dem das einzelne S_i gebildet ist, sondern alle S_i sind zu dem Zeitpunkt eingezeichnet, in dem die Bildung von S_m beendet ist. Für die Bildung der S_i können entweder die F_i und G_i entsprechend (1') oder die A_i und D_i entsprechend (1) verwendet werden. Im Prinzip ist dies bei Rechenzeiten größer als 3τ unwesentlich, da dann die Negation des zuletzt gebildeten C_i die Rechenzeit bestimmt, alle anderen Größen sind vorher gebildet worden.

Nicht erklärt sind bisher die Größen C_0 und U der Abbildung 2. In der Regel ist C_0 gleich Null, aber bei Mehrwort-Additionen, bei Subtraktionen wird auch der Wert $C_0 = L$ benötigt, so daß die Einführung dieser Größe gerechtfertigt ist. C_{25} wird zur Bildung der S_i nicht benötigt, geht aber bei Mehrwort-Additionen in C_0 über und wird sonst zusammen mit C_{24} zur Kontrolle der Richtigkeit der arithmetischen Operation verwendet.

In Abb. 2 sind unter den Kästen der F_i, G_i -Funktionen die notwendigen FAN-OUT-Zahlen angegeben, links für die F -, rechts für die G -Funktionen. Man sieht, daß die FAN-OUT-Zahlen die FAN-IN-Zahlen teilweise erheblich übersteigen, so daß es einfach von der Elektronik abhängt, ob Gl. (1') oder Gl. (1) oder eine weitere Abänderung von (1) für die Bildung der S_i zweckmäßigerweise benutzt wird.

Hier werden zur Bildung der S_i die F_i und G_i nach (1') verwendet, um die Zahlen in (5''') konstant zu halten. Auch werden außer im letzten Kapitel keine Annahmen über die maximale FAN-OUT-Zahl gemacht, d.h. man setzt diese Zahl hinreichend groß an. Dies ist aber in Übereinstimmung mit der zitierten Literatur [1], [2], [3].

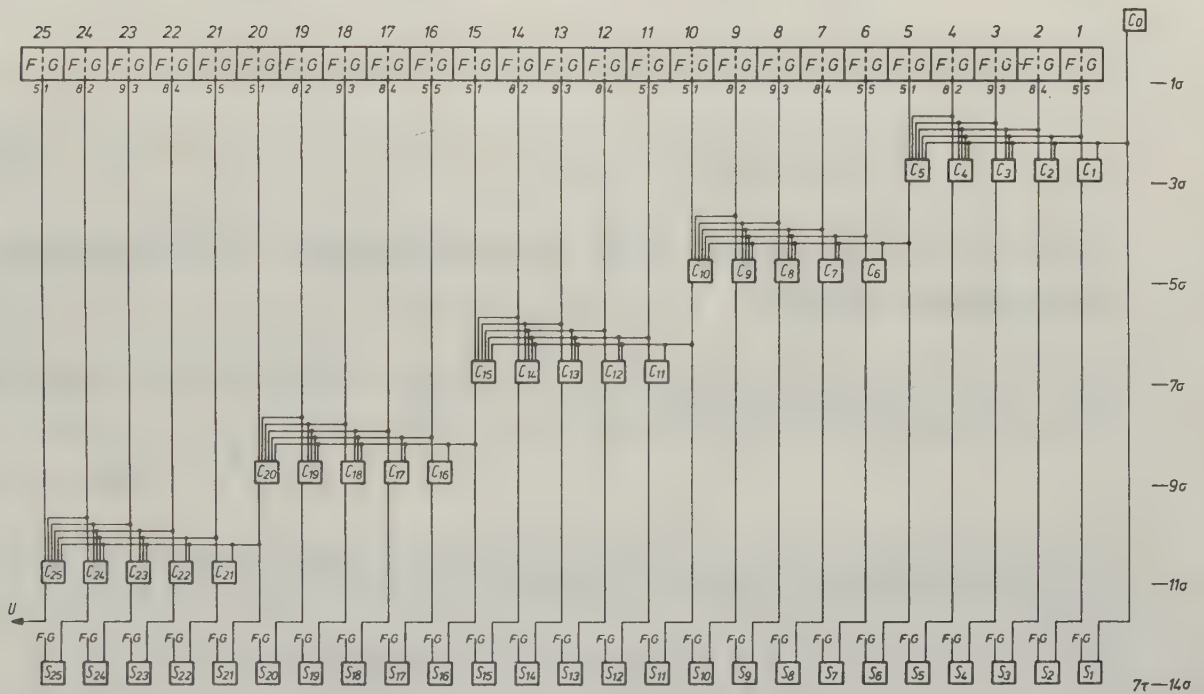


Abb. 2. Stufenaddierer aus 5x5-Bit, n=5

Hilfsfunktionen 1. Art

Es werden wieder wie vorher $n = 5$ benutzt und die Gleichungen für C_{11} , C_{12} als Funktionen von C_5 aufgestellt.

$$C_{11} = G_{11} + F_{11}G_{10} + F_{11}F_{10}G_9 + F_{11}F_{10}F_9G_8 + F_{11}F_{10}F_9F_8G_7 + F_{11}F_{10}F_9F_8F_7G_6 + \\ + F_{11}F_{10}F_9F_8F_7F_6C_5$$

$$C_{12} = G_{12} + F_{12}G_{11} + F_{12}F_{11}G_{10} + F_{12}F_{11}F_{10}G_9 + F_{12}F_{11}F_{10}F_9G_8 + \\ + F_{12}F_{11}F_{10}F_9F_8G_7 + F_{12}F_{11}F_{10}F_9F_8F_7G_6 + F_{12}F_{11}F_{10}F_9F_8F_7F_6C_5$$

Diese Gleichungen können durch Zusammenfassung und Ausklammerung umgeschrieben werden:

$$C_{11} = (G_{11} + F_{11}G_{10} + F_{11}F_{10}G_9 + F_{11}F_{10}F_9G_8 + F_{11}F_{10}F_9F_8G_7 + F_{11}F_{10}F_9F_8F_7G_6) + \\ + (F_{11}F_{10}F_9F_8F_7F_6)C_5 \quad (10)$$

$$C_{12} = G_{12} + F_{12}(G_{11} + F_{11}G_{10} + F_{11}F_{10}G_9 + F_{11}F_{10}F_9G_8 + F_{11}F_{10}F_9F_8G_7 + \\ + F_{11}F_{10}F_9F_8F_7G_6) + F_{12}(F_{11}F_{10}F_9F_8F_7F_6)C_5$$

Es ist zu sehen, daß auch in den entsprechenden Gleichungen für C_{13} , C_{14} und C_{15} die gleichen Faktoren ausklammerbar wären. Es ist weiter zu sehen, daß die Klammerinhalte die maximale FAN-IN-Zahl von 6 gerade ausschöpfen, größere Klammern seien zunächst nicht zulässig. Zeitlich können die Klammerinhalte gleichzeitig mit C_5 gebildet werden, da sie nur aus Elementen bestehen, die zur gleichen Zeit 10 zur Verfügung standen, zu der die für die Bildung von C_5 benötigten Elemente bereitstanden. Das bedeutet aber, daß C_{11} , C_{12} , ..., C_{15} zur gleichen Zeit gebildet werden kann wie C_6 , ..., C_{10} . Diese Klammerinhalte sollen Hilfsfunktionen genannt werden und es soll gelten

$$G_s^r = G_r + F_r G_{r-1} + F_r F_{r-1} G_{r-2} + \dots + F_r F_{r-1} \dots F_{s+1} G_s \quad ,$$

die eine Klammer in (10) hätte also die Bezeichnung G_6^{11} . F_s^r bedeutet entsprechend das Produkt $F_r F_{r-1} \dots F_{s+1} F_s$, das Beispiel in (10) wäre F_6^{11} . Hilfsfunktionen sollen immer Hilfsfunktionen 1. Art genannt werden, wenn Paare G_s^r und F_s^r vorliegen und für die Differenz der Indizes gilt $r - s \leq n$. Der Name 1. Art rührt daher, daß diese Hilfsfunktionen in der Zeit $1 \cdot \tau$ erzeugt werden können, im Gegensatz zu später zu besprechenden Hilfsfunktionen 2. und höherer Art, für deren Bildung 2 oder mehr τ benötigt werden. Dann darf auch $r - s > n$ sein.

Für die G_s^r gelten ähnliche Bildungsgesetze wie für die C_i , sie werden wie diese aus maximal n Paaren von F_t, G_t -Funktionen gebildet, wobei $r \geq t \geq s + 1$ gilt und der Bereich zwischen r und s lückenlos besetzt sein muß. Außerdem geht in G_s^r abschließend noch G_s ein, so wie in C_i abschließend ein anderes C eingeht. Das zu G_s gehörende F_s wird in F_s^r verwendet. Man kann die Eigenschaft, die maximale FAN-IN-Zahl sei $n + 1$, für Hilfsfunktionen 1. Art so umformulieren: In Hilfsfkt.-Paare 1. Art dürfen maximal $n + 1$ Paare von Funktionen F_i, G_i eingehen, wobei die Folge der Indizes lückenlos sein muß.

Schreibt man die Gl. (10) mit Hilfe dieser neuen Bezeichnungen, so erhält man

$$C_{11} = G_6^{11} + F_6^{11} C_5$$

$$C_{12} = G_{12} + F_{12} G_6^{11} + F_{12} F_6^{11} C_5$$

und man kann weiter bilden

$$C_{13} = G_{13} + F_{13} G_{12} + F_{13} F_{12} G_6^{11} + F_{13} F_{12} F_6^{11} C_5$$

$$C_{14} = G_{14} + F_{14} G_{13} + F_{14} F_{13} G_{12} + F_{14} F_{13} F_{12} G_6^{11} + F_{14} F_{13} F_{12} F_6^{11} C_5$$

$$C_{15} = G_{15} + F_{15} G_{14} + F_{15} F_{14} G_{13} + F_{15} F_{14} F_{13} G_{12} + F_{15} F_{14} F_{13} F_{12} G_6^{11} + \\ + F_{15} F_{14} F_{13} F_{12} F_6^{11} C_5 \quad .$$

Man sieht der Gleichung für C_{15} an, daß wieder die maximale FAN-IN-Zahl erreicht ist, C_{16} kann auf diese Weise nicht mehr gebildet werden. Es kann aber zur gleichen Zeit, in der F_6^{11} und G_6^{11} gebildet werden, ebenfalls das Paar F_{12}^{16} und G_{12}^{16} gebildet werden und hier folgt für C_{16}

$$C_{16} = G_{12}^{16} + F_{12}^{16}G_6^{11} + F_{12}^{16}F_6^{11}C_5 \quad (11)$$

und analog wie oben könnte sofort C_{17} , C_{18} und C_{19} gebildet werden, da mit jedem C_i die FAN-IN-Zahl um 1 steigt, C_{16} aber ein FAN-IN von 3 hat.

Wenn man so weiter verfährt, müßte man zur Bildung von C_{20} bereits ein neues Paar F_{17}^{20} und G_{17}^{20} erstellen. C_{20} hätte dann ein FAN-IN von 4 (z.B. aus $F_{17}^{20}, F_{12}^{16}, F_6^{11}$ und C_5), so daß nur noch C_{21} und C_{22} gebildet werden könnten, d.h. die jedem Hilfsfunktionspaar zuordbaren Carrys C_i nehmen bei jedem neuen Funktionspaar 1. Art um 1 ab. Das letzte Carry, das so gebildet werden könnte, wäre C_{29} .

Würde man dagegen F_{12}^{16} und G_{12}^{16} wie in (11) nur für die Bildung von C_{16} benutzen und dann ein volles Hilfsfunktionspaar F_{12}^{17}, G_{12}^{17} bilden, - voll, weil die maximale FAN-IN-Zahl ausgenutzt wird -, so könnte mit diesem Funktionspaar auch noch C_{20} gebildet werden. Für C_{21} und für C_{22} wäre dann wieder ein Hilfsfunktionspaar zusätzlich zu bilden, etwa F_{18}^{21}, G_{18}^{21} . Anschließend würde wieder das volle Paar F_{18}^{23}, G_{18}^{23} gebildet werden, usw.. Das Carry mit dem höchsten Index wäre dann nicht C_{29} , sondern C_{35} .

Dies folgt aber auch aus folgendem Konstruktionsverfahren:

Man bilde n Hilfsfunktionen 1. Art aus je $n + 1$ Funktionspaaren F_i, G_i . Jede dieser Hilfsfunktionen soll, wie schon oben geschehen, eine volle Hilfsfunktion genannt werden, da die maximale FAN-IN-Zahl ausgenutzt wird. Durch diese Hilfsfunktionen werden alle F_i und G_i mit $i = n + 1$, hier 6, bis $n + n(n+1)$, hier 35, erfaßt. Mit diesen vollen Hilfsfunktionen bilde man so viele Carrys, wie es unter Einhaltung der maximalen FAN-IN-Zahl bei den C_i möglich ist. Für die verbleibenden Carrys bilde man weitere Hilfsfunktionen 1. Art.

Für diese weiteren Hilfsfunktionen 1. Art sind in der Regel mehrere Möglichkeiten gegeben, ihre Bildung ist nicht mehr eindeutig, sie wird erst dann wieder eindeutig, wenn ein weiteres Kriterium, z.B. geringster Elektronikaufwand, dazukommt. Dieses Problem soll später betrachtet werden.

Die Zahl der aus den vollen Hilfsfunktionen 1. Art direkt bildbaren Carry läßt sich wie folgt übersehen:

Nummeriert man die vollen Hilfsfunktionen 1. Art mit $k = 1$ bis n , wobei k in der gleichen Richtung wie die Indizes r oder s wachsen soll, so läßt sich sagen, es gibt für jedes k ein C_1 , das nur aus k vollen Hilfsfunktionen 1. Art (und dem abschließendem Carry) gebildet ist. Ist $k < n$, dann kann auch C_{i+1} unter Hinzuziehung von F_{i+1}, G_{i+1} direkt gebildet werden, war $k < n - 1$, dann gilt dies auch für C_{i+2} unter der zusätzlichen Verwendung von F_{i+2} und G_{i+2} , usw., bis die zulässige Höchstzahl n für die Funktionenpaare eines Carry erreicht ist. Mit der k -ten Hilfsfunktion 1. Art können also $n + 1 - k$ Carry direkt gebildet werden. Für $k = n$ kann nur C_{n^2+2n} allein gebildet werden.

Auch bei dieser Anordnung läßt sich sagen, daß die Zahl der direkt bildbaren Carry mit jedem weiteren Hilfsfunktionspaar 1. Art um 1 abnimmt, aber es existieren für $k < n$ zu jeder vollen Hilfsfunktion 1. Art genau k Carrys, die durch weitere nicht volle Hilfsfunktionen 1. Art gebildet werden müssen. Das kann aber in der gleichen Zeit geschehen, in der die übrigen Carry gebildet werden.

Man kann den Sachverhalt auch so ausdrücken:

Mit Hilfsfunktionen 1. Art läßt sich eine Gruppe von n^2 Carry gleichzeitig mit n weiteren Carry, die keine Hilfsfunktionen benötigen, bilden. Die Stufenlänge beträgt also $n^2 + n$, die Länge des ganzen Addierers, der in der Gesamtrechnzeit von 4τ arbeiten kann, ist

$$n^2 + 2n \quad . \quad (12)$$

Da moderne Elektronik maximale FAN-IN von 8 zuläßt, ergibt sich n zu 7 und der höchste Carry-Index zu $n^2 + 2n = 63$. Ein solcher Addierer kann also in der Zeit 4τ die Addition durchführen, für

die ein einfacher Addierer 64τ benötigt.

Zur Demonstration der Hilfsfunktionen 1. Art soll ein 25-Bit-Addierer bei einem maximalem FAN-IN von 5 in Abb. 3 betrachtet werden. Die Länge der 1. Stufe ist gleich $n = 4$, der höchste in der 2. Stufe erreichbare Carry-Index gehört nach (12) zu C_{24} , die Bildung von C_{25} kostet ein weiteres τ , so daß die Gesamtrechnzeit 5τ wäre. Der Abbildung 3 ist zu entnehmen, daß C_1 bis C_4 in der Zeit 3σ gebildet werden und daß die Konfiguration der Gruppe C_5 bis C_8 die gleiche wie die der Gruppe C_1 bis C_4 ist, nur müssen die F_1 und G_1 für i von 5 bis 8 noch zur Bildung von F_5^9 und G_5^9 , dem 1. Hilfsfunktionspaar 1. Art, beitragen. Die weiteren vollen Hilfsfunktionspaare 1. Art haben die Indizes $\begin{smallmatrix} 14 & 19 & 24 \\ 10 & 15 & 20 \end{smallmatrix}$. Die vollen Hilfsfunktionspaare sind in Abb. 3 durch Kästen mit schwarzen Ecken gekennzeichnet.

Vom ersten Funktionenpaar $\begin{smallmatrix} 9 \\ 5 \end{smallmatrix}$ kann $C_9, C_{10}, C_{11}, C_{12}$ direkt, C_{13} mit einem weiteren Funktionenpaar 1. Art gebildet werden. Es ist aber im Elektronikaufwand günstiger, statt des Paares mit den Indizes $\begin{smallmatrix} 13 \\ 10 \end{smallmatrix}$ das Paar mit den Indizes $\begin{smallmatrix} 12 \\ 10 \end{smallmatrix}$ zu bilden und dieses Funktionspaar zur Bildung von C_{12} und C_{13} heranzuziehen. Fast der gleiche Sachverhalt liegt bei C_{18} und C_{17} vor, nur muß hier notwendigerweise zur Bildung von C_{17} F_{15}^{17} und G_{15}^{17} gebildet werden, diese Hilfsfunktionen genügen aber auch zur Bildung von C_{18} . Von C_{19} bis C_{23} zeigt sich das typische Bild der letzten Gruppe eines Addierers mit Hilfsfunktionen 1. Art, jedes C_i mit Ausnahme von C_{20} benötigt ein eigenes Hilfsfunktionspaar 1. Art.

Man sieht, daß innerhalb einer solchen Stufe, die $n^2 + n$ Carry bildet und Hilfsfunktionen 1. Art benutzt, der Aufwand an Elektronik von rechts nach links, oder mit wachsendem Carry-Index steigt. Dies wird später ausgenutzt werden, wenn wie hier bei der Bildung von C_{25} eine weitere Stufe benützt werden muß, die zusätzliche Zeit kostet, aber auch Gelegenheit zu erheblichen Einsparungen an Elektronik bietet.

Für diesen 25-Bit-Addierer, der das Ergebnis in 5τ liefert, sei - vorbehaltlich einer späteren Angabe des Berechnungsverfahrens -

die benötigte Elektronik für die Carry-Bildung mit

$$\begin{array}{llll} 34, & 106, & 470 & \text{angeben, woraus sich mit } (5''') \\ 84, & 206, & 820 & 5\tau \end{array} \quad (13)$$

für den ganzen Addierer ergibt.

Die Vergleichstabelle sieht dann wie folgt aus, fügt man auch noch die entsprechenden Werte für einen bei 25 Bit abgebrochenen Addierer hinzu, der mit $n = 5$ aufgebaut ist:

75,	425,	3950	3τ	nicht erreichbar	(6')
80,	204,	841	4τ	Hilfsfkt. 1. Art, $n = 5$	(14)
84,	206,	820	5τ	Hilfsfkt. 1. Art, $n = 4$	(13)
75,	175,	700	7τ	5×5 Gruppen, $n = 5$	(9)
75,	125,	450	26τ	langsamer Addierer	(6'')

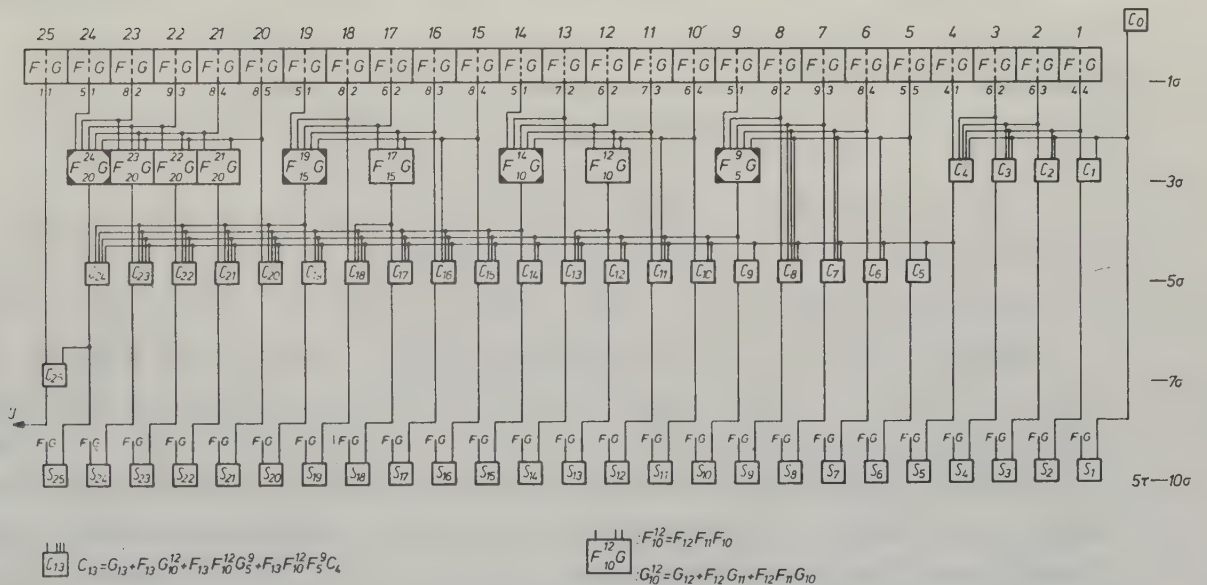


Abb. 3. Addierer, systematisch mit Hilfsfunktionen 1. Art, $n=4$

Die Verwendung von Hilfsfunktionen 1. Art erhöht hier die Zahl der OR-Glieder um etwa 10% gegenüber den anderen Addierer-Formen. Die Zahl der AND-Glieder ist gegen (6'') um $\frac{2}{3}$ vermehrt, aber nur noch um 40% gegenüber (9), dem steht aber ein Gewinn in der Rechenzeit von $2 - 3\tau$ gegenüber, also fast nochmals eine Halbierung der Rechenzeit. Die Zunahme der Dioden liegt in Größenordnung von 80% gegenüber (6'') und beträgt 20% gegenüber (9).

Bisher wurde beschrieben, wie eine Rechenstufe unter Gebrauch von Hilfsfunktionen 1. Art aufgebaut werden kann. Freiheit besteht noch in einem gewissen Umfange in der Wahl der nicht vollen Hilfsfunktionen. Deren Festlegung soll erst nach der Einführung der Hilfsfunktionen 2. und höherer Art geschehen.

Hilfsfunktionen 2. Art

Es war gezeigt worden, daß es ein maximales FAN-IN von $n + 1$ gestattet, mit Hilfsfunktionen 1. Art die Addition von $n^2 + 2n$ Bit in der Gesamtrechnenzeit von 4τ zu erledigen. Ist die Bitzahl des Addierers m größer als $n^2 + 2n$, so vergrößert sich die Gesamtrechnenzeit mindestens auf 5τ und die Carry-Bildung erfolgt in mindestens 3 Stufen. C_{n^2+2n} dient als Carry, mit dem die Carry der 3. Stufe gebildet werden.

Zunächst aber benötigen die Carry C_{n^2+2n+1} bis C_{n^2+3n} ebenso wenig Hilfsfunktionen, wie C_1 bis C_n und C_{n+1} bis C_{2n} sie benötigten. Anschließend können, beginnend mit C_{n^2+3n+1} , n^2 weitere Carry gebildet werden, die nur Hilfsfunktionen 1. Art benötigen und die analog den Carryfunktionen der 2. Stufe aufgebaut sind.

Die Notwendigkeit, Hilfsfunktionen 2. Art zu gebrauchen, liegt aber erst vor, wenn die Addierer-Bitzahl m größer als $2n^2 + 3n$ ist - für $n = 5$ wäre dies $m > 65$ - und wenn der Addierer in der Gesamtrechnenzeit 5τ arbeiten soll. Stünde dem Addierer mehr Zeit zur Verfügung, so wäre es naheliegend, in Abhängigkeit vom Carry C_{2n^2+3n} eine 4. Stufe von $n^2 + n$ Elementen zu bilden, die ebenfalls nur maximal Hilfsfunktionen 1. Art benötigten.

Der Vorgang der Einführung und der Bildung von Hilfsfunktionen 2. Art ist der gleiche wie der bei der Bildung der Hilfsfunktionen 1. Art. Betrachtet man die F_i und G_i als Hilfsfunktionen 0-ter Art, so kann man sagen, daß Hilfsfunktionen 2. Art den gleichen Aufbau wie Hilfsfunktionen 1. Art besitzen, nur muß unter den maximal $n + 1$ Funktionspaaren, aus denen die Funktionen 2. Art gebildet sein können, sich mindestens ein Paar von der 1. Art befinden. Funktionen 0-ter Art sollen ausdrücklich weiter zugelassen werden. Daraus folgt, daß $r - s > n$ sein darf und daß diese Funktionen 2. Art frühestens nach der Zeit von 5σ gebildet sein können, oder 2τ nach der Bildung der F_i, G_i .

Fragt man wieder nach dem größten i , dessen C_i noch mit Hilfsfunktionen 2. Art gebildet werden kann, so sieht man, daß in dieses C_i n Paare von Hilfsfunktionen 2. Art und das abschließende C eingehen können. Jede Hilfsfunktion 2. Art kann aber aus $n + 1$ Hilfs-

funktionenpaaren 1. Art gebildet sein, die wiederum jede aus $n + 1$ Hilfsfunktionspaaren 0-ter Art gebildet sind. Die 3. Stufe, in der zum erstenmal Hilfsfunktionen 2. Art verwendet werden können, umfaßt also $n \cdot (n+1)(n+1)$ Bit, das abschließende Carry der 2. Stufe ist C_{n^2+2n} , so daß das gesuchte $i = n^3 + 3n^2 + 3n$ ist. Mit Hilfsfunktionen 2. Art lassen sich in einer Gesamtrechnzeit von 5τ genau $n^3 + 3n^2 + 3n$ Bit addieren. Für $n = 5$ sind das 215 Bit.

Für $n = 2$ sind dies aber nur 26 Bit, und wenn es zweckmäßig sein sollte, das maximale FAN-IN möglichst klein zu machen, dann können auch Hilfsfunktionen p -ter Art mit $p \geq 2$ eine Rolle spielen.

Das Konstruktionsverfahren eines Addierers mit Hilfsfunktionen 2. Art ist ebenfalls analog dem Verfahren bei Funktionen 1. Art. Über der Reihe der Carry C_1 der 3. Stufe mit $i = n^2 + 2n + 1$ bis $i = n^3 + 3n^2 + 3n$ bilde man an jeder $(n+1)^2$ -ten Stelle ein volles Hilfsfunktionenpaar 2. Art, das sind genau n Funktionenpaare mit der Index-Differenz $r - s = (n+1)^2 - 1$, und in einer darüber liegenden Reihe $n(n+1)$ Hilfsfunktionenpaare 1. Art, die alle zur Bildung der vollen Funktionspaare 2. Art notwendig sind. Dabei kommt jedes $n+1$ -te Funktionspaar 1. Art über einem der Funktionspaare 2. Art zu liegen. Dann bilde man aus diesen beiden Sorten von Hilfsfunktionenpaaren alle Carry, die direkt bildbar sind, wenn die maximale FAN-IN-Zahl für die Carry eingehalten wird.

Anschließend werden weitere, nicht mehr volle, Hilfsfunktionen 1. und 2. Art aufgestellt, um die restlichen Carry bilden zu können. Die Auswahl dieser weiteren Hilfsfunktionen ist wieder nicht eindeutig vorgeschrieben, eindeutige Vorschriften benötigen weitere Kriterien.

Der Beweis, daß diese weiteren Hilfsfunktionen für jede Carryfunktion gebildet werden können, soll im nächsten Abschnitt nachgeholt werden, vorher soll noch ein Addierer mit Hilfsfunktionen 2. Art betrachtet werden.

Für diesen Addierer wurde $n = 2$ gewählt, um ihn überschaubar zu machen (Abb. 4). Aus $n = 2$ folgen die Grenzen der 3. Stufe zu $n^2 + 2n + 1 = 9$ und $n^3 + 3n^2 + 3n = 26$. Die zwei vollen Funktionspaare 2. Art haben die Indizes 9^{17} und 26^{26} , die vollen Funktionspaare 1. Art benötigen die Indizes 11^{18} , 14^{17} , 20^{23} und 26^{24} . Mit diesen vollen Hilfsfunktionen lassen sich direkt nur C_9 , C_{10} , C_{11} , C_{12} , dann C_{14} , dann C_{17} , C_{18} , dann C_{20} und am Schluß C_{26} bilden, das sind genau die Hälfte der Carry dieser Stufe.

Die Lücke C_{13} könnte mit Funktionen 1. Art, deren Indizes 13^{13} sind, gebildet werden, Elektronikeinsparungen sprechen für Funktionen 2. Art mit den Indizes 9^{12} , mit denen dann C_{12} und C_{13} gebildet werden. Zur Bildung von C_{15} und C_{16} lassen sich die Funktionen 2. Art, deren Indizes 15^{15} sind, aufstellen. Die nächste Lücke ist C_{19} , das nur mit den Hilfsfunktionen 1. Art, deren Indizes 19^{19} sind, gebildet werden kann. In den letzten Gruppen von C_{21} bis C_{22} benötigt jedes C_i ein Paar Hilfsfunktionen 2. Art mit den Indizes 1^{18} , außerdem wird ein Hilfsfunktionenpaar 1. Art mit den Indizes 25^{24} benötigt.

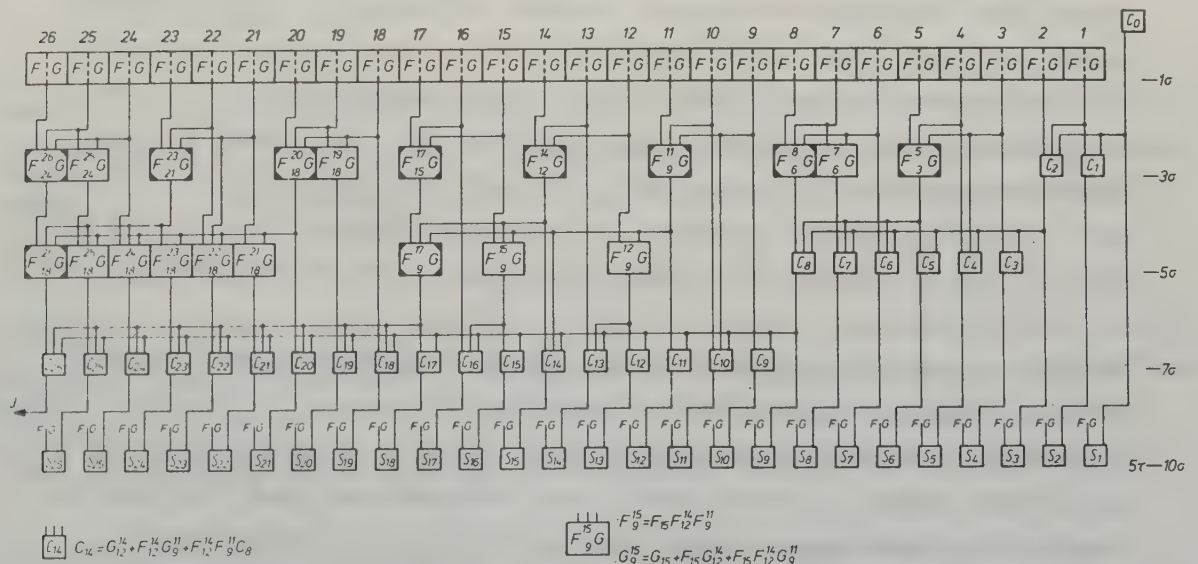


Abb. 4. Addierer, systematisch mit Hilfsfunktionen 2. Art, $n=2$

Hilfsfunktionenpaare p-ter Art

Der Übergang von Hilfsfunktionen 1. Art zu solchen 2. Art läßt sich verallgemeinern, um die Eigenschaften von Hilfsfunktionen p-ter Art kennenzulernen.

Ein Hilfsfunktionspaar soll ein Hilfsfunktionspaar p-ter Art genannt werden, wenn unter den zu seiner Bildung verwendeten Funktionen mindestens ein Hilfsfunktionspaar p-1 - ter Art verwendet wird.

Ein Hilfsfunktionenpaar p-ter Art, daß aus genau $n + 1$ vollen Hilfsfunktionenpaaren p-1 - ter Art gebildet wird, soll ein volles Hilfsfunktionspaar p-ter Art genannt werden.

Daraus folgt, daß der Bereich der Indizes der F_1, G_1 , die zur Bildung eines vollen Hilfsfunktionspaares p-ter Art direkt oder indirekt benötigt werden, um den Faktor $n + 1$ größer ist als der entsprechende Bereich für Funktionen p-1 - ter Art.

Entsprechend ist auch die Länge der Stufe, auf der Funktionen p-ter Art erstmals verwendet werden können, $n + 1$ mal länger als die Stufe, auf der Funktionen p-1 - ter Art erstmals verwendet werden können. Denn jede Stufenlänge erfordert jeweils n solcher Funktionspaare. Die Carry-Stufe, zu deren Bildung Funktionen p-ter Art erstmals verwendet werden können, soll die $p+1$ - Stufe genannt werden, in Übereinstimmung damit, daß für die 2. Stufe erstmals Hilfsfunktionen 1. Art verwendet wurden.

Da die F_1 und G_1 als Funktionen 0-ter Art betrachtet werden müssen, die bei einem maximalem FAN-IN von $n + 1$, in der 1. Stufe n Carry gleichzeitig bilden können, ergibt sich für die 2. Stufe mit Funktionen 1. Art $n(n+1)^1$ als Stufenlänge und für die $p+1$ - te Stufe mit Funktionen p-ter Art $n(n+1)^p$ als Stufenlänge.

Als Länge eines Addierers, der mit Funktionen p-ter Art in $p+1$ Stufen gebildet werden kann, ergibt sich
$$\sum_{i=0}^{i=p} n(n+1)^i = (n+1)^{p+1} - 1.$$

Die Gesamtrechnenzeit, die gebraucht wird, wenn der Addierer $p + 1$ Stufen enthält, ergibt sich zu $(p+3)\tau$. Dies ist in Übereinstimmung mit der früher gefundenen Formel $(k+2)\tau$, wobei k die Zahl der Carry-Stufen war, es ist also $p + 1 = k$.

Aus der folgenden Aufstellung sieht man die Übereinstimmung der Formel-Werte mit den früher gefundenen Werten für $p = 0, 1, 2$

p	k	$n(n+1)^p$	$(n+1)^{p+1}-1$	$(k+2) \cdot \tau$
0	1	n	n	3τ
1	2	n^2+n	n^2+2n	4τ
2	3	n^3+2n^2+n	n^3+3n^2+3n	5τ
3	4	$n^4+3n^3+3n^2+n$	$n^4+4n^3+6n^2+4n$	6τ

Das verallgemeinerte Konstruktionsprinzip für einen Addierer sieht so aus: Die $p+1$ - te Stufe besitzt eine Stufenlänge von $n(n+1)^p$ Carry. Von rechts in der Stufe beginnend bilde man über jedem $(n+1)^p$ - ten Carry-Index ein volles Hilfsfunktionenpaar p -ter Art. Dies ist genau n -mal möglich und die Index-Differenz dieser Paare $r - s$ ist gleich $(n+1)^p-1$.

Dann bilde man in einer darüberliegenden Reihe zu jedem $(n+1)^{p-1}$ -ten Carry-Index ein volles Hilfsfunktionenpaar $p-1$ - ter Art. Dies ist genau $n(n+1)$ -mal möglich und die Index-Differenz dieser Paare ist $r - s$ gleich $(n+1)^{p-1}-1$.

Dieses Verfahren setze man fort, bis in einer obersten Reihe, wieder von rechts beginnend, zu jedem $n+1$ - ten Carry-Index ein volles Hilfsfunktionenpaar 1. Art gebildet ist. Darüber befindet sich dann die Reihe der F_1 und G_1 , also die Funktionen 0-ter Art.

Aus dieser Anordnung ergibt sich, daß über jedem vollen Hilfsfunktionenpaar j -ter Art alle vollen Hilfsfunktionspaare i -ter Art mit $i < j$ vorhanden sind, speziell ist über dem letzten linken Bit der Stufe mit dem letzten der Hilfsfunktionspaare p -ter Art auch sofort die ganze Skala der Hilfsfunktionspaare von geringerer

als p-ter Art vorhanden, dies Carry kann direkt gebildet werden.

Nach der Aufstellung aller dieser vollen Funktionspaare von der p-ten bis zur 1. Art bilde man alle Carry, die unter Einhaltung der maximalen FAN-IN-Zahl aus diesen vollen Funktionen und den F_i und G_i direkt gebildet werden können.

Die Bildung von weiteren Hilfsfunktionen, um die restlichen Carry bilden zu können, ist nicht eindeutig bestimmt, es können weitere Bildungskriterien hinzutreten.

Man kann aber einsehen, daß es immer möglich ist, Hilfsfunktionen für jedes Carry zu finden, das nicht direkt aus den vollen Hilfsfunktionen gebildet werden kann. Es sei C_i ein Carry, das nicht direkt gebildet werden kann, und es sei der ungünstigste Fall angenommen, daß kein volles Hilfsfunktionenpaar existiere, dessen oberer Index $r = i$ sei.

Dann muß rechts vom Index i ein volles Hilfsfunktionenpaar 1. Art $F_{s_1}^{r_1}, G_{s_1}^{r_1}$ existieren, so daß $i - r_1$ kleiner als $n + 1$ ist, da zu jedem $n+1$ - ten Carry-Index ein volles Hilfsfunktionenpaar 1. Art gebildet worden war.

Also kann ein neues Hilfsfunktionenpaar 1. Art $F_{r_1+1}^i, G_{r_1+1}^i$ gebildet werden.

Zwischen dem Index i und dem nächsten rechtsgelegenen vollen Hilfsfunktionenpaar 2. Art $F_{s_2}^{r_2}, G_{s_2}^{r_2}$ können maximal nur n volle Hilfsfunktionenpaare 1. Art liegen. Aus diesen letzteren und dem oben gebildeten Paar $F_{r_1+1}^i, G_{r_1+1}^i$ kann ein weiteres Paar 2. Art $F_{r_2+1}^i, G_{r_2+1}^i$ gebildet werden, die maximale FAN-IN-Zahl wird nicht überschritten.

Diese Bildung von Hilfsfunktionen kann fortgesetzt werden, bis ein Funktionspaar p-ter Art mit dem oberen Index $r = i$ entstanden ist. Zusammen mit weiteren maximal $n-1$ Hilfsfunktionenpaaren p-ter Art kann dann C_i gebildet werden.

In der Regel wird man aber nicht so vorgehen, sondern danach trachten, diese weiteren Hilfsfunktionen so einzuführen, daß mit möglichst wenig Hilfsfunktionen möglichst viele Carry erzeugt werden können.

Gesamtrechenzeit als Funktion von m und n

Die bisher angestellten Überlegungen reichen aus, um die Frage beantworten zu können, welche Gesamtrechenzeit in Vielfachen von τ erreicht und welche Art von Hilfsfunktionen benötigt wird, wenn die Zahl der Bit des Addierers m gegeben ist und wenn aus der verwendeten Elektronik ein maximales FAN-IN folgt, das um 1 vermindert, die Größe n ergibt. Dies geschieht am übersichtlichsten mit der Abbildung 5, die die in den vorangehenden Abschnitten ermittelten Formeln zusammenzufassen versucht.

In der rechten Spalte dieser Abb. 5 ist die maximale FAN-IN-Zahl $n + 1$, links daneben die Zahl n selbst für eine Reihe passender Werte eingetragen. Ist ein bestimmtes maximales FAN-IN vorgegeben, so ist dadurch in Abb. 5 eine bestimmte Zeile ausgewählt. In dieser Zeile suche man dann die Zahl q, die die kleinste Zahl dieser Zeile mit $q \geq m$ sein soll.

Dieses q bestimmt eine Spalte, in der in der Zeile 8' die Gesamtrechenzeit gefunden werden kann.

In der durch q bestimmten Spalte und in den rechts davon gelegenen Spalten kann in den Zeilen 1' bis 7' ein grobes Block-Schema des entsprechenden Addierers entnommen werden, wobei die Art der verwendeten Hilfsfunktionen in den Kästen sofort abgelesen werden kann.

Verglichen mit der Abb. 2,3 und 4 ist dieses Block-Schema grob, da die Kästen für die Bildung der F_1 und G_1 sowie der S_1 nicht eingezeichnet sind, darüber hinaus sind die Kästen für die Bildung der Carry-Funktionen in jeder Stufe zu einem Block zusammengefaßt. Das letztere gilt auch für alle Hilfsfunktionen 1.,2.,... Art. Diese Blocks sind nur noch in Gruppen unterteilt und diese Gruppen einer Stufe sind dadurch untereinander verschieden, daß die Carry, die zu verschiedenen Gruppen gehören, verschiedene Arten von Hilfsfunktionen zu ihrer Bildung benötigen. In der Zeile 9' ist die Anzahl der Bit jeder Gruppe als Funktion von n aufgetragen, man sieht,

Frage entsteht, welches FAN-IN muß die Elektronik besitzen. Dies findet man in Abb. 5, wenn man in der am weitesten links liegenden Spalte, die zur vorliegenden vorgegebenen Gesamtrechnzeit gehört, wieder q mit $q \geq m$ sucht. Das in dieser Zeile rechts angegebene maximale FAN-IN ist notwendig.

Zum Schluß soll noch erwähnt werden, daß alle in den vorhergehenden Abschnitten erwähnten speziellen numerischen Resultate in der Abb. 5 wiederzufinden sind.

Reduzierung der Elektronik

In den vorangehenden Abschnitten war eine Systematik für den Aufbau eines Addierers entwickelt worden, wobei öfter auf die Möglichkeit hingewiesen wurde, Elektronik einsparen zu können. Andererseits kann das Bestreben, Elektronik einzusparen, dazu dienen, diejenigen Teile des Addierers eindeutig zu bestimmen, bei denen die bisher angegebene Systematik noch Freiheiten gelassen hatte. Sind die für diesen Zweck notwendigen Hilfsmittel aber erst entwickelt, so wird sich zeigen, daß noch drei weitere Probleme mit ihnen gelöst werden können.

Einmal fällt in der Regel die Addierer-Bitzahl m nicht mit dem Ende einer Stufe oder dem Ende einer Gruppe in einer Stufe zusammen. Dann wird die letzte Gruppe vorzeitig abgebrochen und es entsteht die Frage, ob an der Spitze des Addierers, für die höchsten Indizes, die systematischen vollen Hilfsfunktionen wirklich die günstigsten im Sinne des Elektronikaufwandes sind.

Bei Addierern, die mehr als 3τ Gesamtrechnenzeit besitzen, werden die Carry der 1. Stufe alle um die Zeit τ vor dem Carry der 2. Stufe gebildet. Prinzipiell ist es aber nur nötig, daß diese Carry der 1. Stufe - mit Ausnahme des letzten Carry dieser Stufe, dem C_n - nicht später als die Carry der letzten Stufe gebildet werden. Für die nächste Stufe gilt dies ebenfalls. Wenn dieser noch eine weitere Stufe folgt, dann muß nur das letzte Carry, das Carry mit dem höchsten Index innerhalb der ersten Stufe sofort gebildet werden, da es in die Carry-Funktionen der weiteren nächsten Stufe eingeht. Die anderen Carry können gleichzeitig mit den Carry der letzten Stufe gebildet werden. Auch hier liegen eine Reihe von Möglichkeiten, Elektronik einzusparen.

Die dritte Möglichkeit, Elektronik zu sparen, liegt vor, wenn bei großem n , $n > 3$, an Stellen, an denen es der systematische Aufbau nicht fordert, Hilfsfunktionspaare eingeführt werden, die den Elektronikaufwand in den Carry-Funktionen so stark mindern, daß der Elektronikaufwand für diese Hilfsfunktionspaare gerechtfertigt ist. In diesem letzten Falle muß aber die Art der verwendeten

Elektronik stärker berücksichtigt werden. Sind OR- und AND-Glieder, wie bisher angenommen, selbständige elektronische Bausteine mit eigenen Verstärkern, dann kann es sinnvoll sein, diese Überlegungen anzustellen. Ist aber die Zahl der Verstärker gleich der Zahl der OR-Glieder und sind die AND-Glieder durch Dioden-Funktionen erzeugt, dann sieht hier der Konstrukteur möglicherweise keinen Vorteil.

Die Elektronik-Tabellen

Das Hilfsmittel, mit dem die vorher aufgeworfenen Fragen gelöst werden sollen, ist eine Tabelle, die angibt, wieviel Elektronik jede einzelne Carry-Funktion und jedes einzelne Hilfsfunktionenpaar besitzt. Die Menge dieser Elektronik ist durch die Zahl der Funktionspaare bestimmt, die in jede dieser Carry-Funktionen oder Hilfsfunktionenpaare eingehen. Im einfachsten Fall ergibt sich bei

$$C_1 = G_1 + F_1 C_{1-1} \quad \text{oder} \quad C_1 = G_s^1 + F_s^1 C_{s-1}$$

daß für die Bildung von C_1 ein OR-Glied, ein AND-Glied und 4 Dioden oder Zuleitungsdrähte benötigt werden.

$$C(2) = 1, 1, 4 \quad (16)$$

wäre das entsprechende Zahlenbeispiel mit der Bezeichnung $C(2)$, wobei 2 die Zahl der Eingänge (1 Carry, 1 Funktionspaar) angibt. Entsprechend findet man für $C(3)$ die Werte 1, 2, 8. $C(3)$ hat genau 1 AND-Glied mehr als $C(2)$ und dadurch hat das OR-Glied einen Eingang mehr. Außerdem hat das AND-Glied selbst 3 Eingänge, so daß die Zahl 8 erklärt ist. Man könnte für die Berechnung der zu $C(1)$ gehörenden Zahlen analoge Formeln wie in (4') bis (4''') aufstellen, praktischer sind Rekursionsformeln, die das Tripel jeder Zeile aus der vorhergehenden entwickelt:

$$\text{Aus} \quad C(a+1) = 1, a, b \quad (17)$$

$$\text{folgt die Zeile} \quad C(a+2) = 1, a+1, b+a+3$$

und die 1. Zeile dieser Rekursionsvorschrift ist (16) mit $a = 1$.

Die Richtigkeit dieser Vorschrift sieht man so ein: eine Carry-Funktion, die a AND-Glieder besitzt, hat an einem dieser AND-Glieder genau $a + 1$ Eingänge. Wird die Zahl der AND-Glieder um 1 Vermehrt, so besitzt dieses neue AND-Glied $a + 2$ Eingänge, außerdem besitzt dieses AND-Glied einen Eingang in das OR-Glied, so daß $a + 3$ neue Dioden oder Verbindungsdrähte notwendig sind.

Das einfachste Hilfsfunktionenpaar hat die Form

$$F_{i-1}^1 = F_i F_{i-1} \quad \text{und} \quad G_{i-1}^1 = G_i + F_i G_{i-1}$$

und diese Form ändert sich nicht, wenn statt der Funktionenpaare 0-ter Art solche höherer Art genommen worden wären. Die Abzählung ergibt, daß zur Darstellung dieser beiden Funktionen 1 OR-Glied, 2 AND-Glieder und 6 Dioden benötigt werden. Die Gl.(16) entsprechende Gleichung lautet

$$FG(2) = 1, 2, 6. \quad (18)$$

Gl.(18) kann als Anfangswert für ein ähnliches Rekursionsverfahren benutzt werden:

$$\text{Aus} \quad FG(a) = 1, a, b \quad (19)$$

$$\text{folgt} \quad FG(a+1) = 1, a+1, b+a+3.$$

Die Richtigkeit sieht man ähnlich wie bei (17) ein. $FG(a)$ hat 2 AND-Glieder mit je a Eingängen. Wird a um 1 vergrößert, so kommt für die G -Funktion ein neues AND-Glied hinzu, daß $a+1$ Eingänge besitzt, selbst einen Eingang in das OR-Glied benötigt und außerdem benötigt die F -Funktion einen Eingang mehr, so daß der gesamte Zuwachs wieder $a + 3$ ist.

Mit diesen angegebenen Formeln läßt sich dann die folgende Tabelle II leicht aufstellen:

$C(2) = 1, 1, 4$	$FG(2) = 1, 2, 6$
$C(3) = 1, 2, 8$	$FG(3) = 1, 3, 11$
$C(4) = 1, 3, 13$	$FG(4) = 1, 4, 17$
$C(5) = 1, 4, 19$	$FG(5) = 1, 5, 24$
$C(6) = 1, 5, 26$	$FG(6) = 1, 6, 32$
$C(7) = 1, 6, 34$	$FG(7) = 1, 7, 41$
$C(8) = 1, 7, 43$	$FG(8) = 1, 8, 51$
$C(9) = 1, 8, 53$	$FG(9) = 1, 9, 62$

Tabelle II

Fixierung nicht voller Hilfsfunktionen

Bei der Beschreibung bestimmter Addierer-Modelle (Abb. 3 und 4) war ausgeführt worden, daß aus Gründen der Elektronikeinsparung die Wahl auf bestimmte Hilfsfunktionspaare gefallen war. An dem Hilfsfunktionspaar 1. Art mit den Indizes $\begin{smallmatrix} 12 \\ 10 \end{smallmatrix}$ der Abb. 3 soll das Vorgehen demonstriert werden. Außer dem Hilfsfunktionspaar sind noch die Carry-Funktionen C_{10} , C_{11} , C_{12} und C_{13} betroffen. Schreibt man diese Funktionen von der Abb. 3 in der gleichen Anordnung ab, so entsteht

$$\begin{array}{ccccccc} & & & & & & F_{10}^{12}G \\ & & & & & & \\ C_{13} & & C_{12} & & C_{11} & & C_{10} \end{array} \quad (20)$$

Jede dieser Funktionen ist aus einer bestimmten in Abb. 3 entnehmbaren Anzahl von anderen Funktionen gebildet, so daß mit Hilfe der in Tabelle II aufgeführten Ausdrücke die Anordnung (20) umgeschrieben werden kann in:

$$\begin{array}{ccccccc} & & & & & & FG(3)_{10}^{12} \\ & & & & & & \\ C(4)_{13} & & C(3) & & C(4) & & C(3)_{10} \end{array} , \quad (21)$$

zur leichteren Identifizierung soll nach Bedarf noch das zugehörige Indexpaar angehängt werden. Bei der Entwicklung war als Alternative der Gebrauch des Hilfsfunktionspaares 1. Art mit den Indizes $\begin{smallmatrix} 13 \\ 10 \end{smallmatrix}$ erwähnt worden. Ändert man (21) dementsprechend ab, so erhält man

$$\begin{array}{ccccccc} & & & & & & FG(4)_{10}^{13} \\ & & & & & & \\ C(3) & & C(5) & & C(4) & & C(3) \end{array} \quad (22)$$

Diese Änderungen sind verständlich, ohne daß man jedesmal zu Änderungen in Abb. 3 schreitet und dann den Übergang zu (22) ausführt. Ein Verschieben von $FG(i)$ um eine Stelle nach links erzeugt $FG(i+1)$, stand unter diesem $FG(i)$ ein $C(j)$, dann steht auch unter $FG(i+1)$ wieder $C(j)$, und sonst steht immer links neben $C(k)$ stets $C(k+1)$.

Daraus wird

$$FG(6)_{16}^{21}$$

$$C(4)_{22} \quad C(3) \quad C(7) \quad C(6) \quad C(5) \quad C(4) \quad C(3)_{16}$$

also ein Übergang von FG(7) nach FG(6) und von C(8) nach C(4).
Schiebt man nochmals, so entsteht

$$FG(5)$$

$$C(5) \quad C(4) \quad C(3) \quad C(6) \quad C(5) \quad C(4) \quad C(3) \quad ,$$

also ein Übergang von FG(6) nach FG(5) und von C(7) nach C(5).
Schiebt man ein drittesmal, entsteht

$$FG(4)$$

(24)

$$C(6) \quad C(5) \quad C(4) \quad C(3)_{19} \quad C(5)_{18} \quad C(4)_{17} \quad C(3)_{16} \quad ,$$

jetzt findet nur noch ein Übergang von FG(5) nach FG(4) statt, die
C(i) bleiben ungeändert, nur in der Reihenfolge sind sie verschoben.

Die nächste Verschiebung

$$FG(3)$$

$$C(7) \quad C(6) \quad C(5) \quad C(4) \quad C(3) \quad C(4) \quad C(3)$$

bringt zwar den Übergang FG(4) nach FG(3) aber auch den Wechsel
von C(5) nach C(7). Tabelle II ergibt:

$$FG(4) \rightarrow FG(3) \quad 0 \quad , \quad -1 \quad , \quad -6$$

$$C(5) \rightarrow C(7) \quad 0 \quad , \quad 2 \quad , \quad 15$$

Das Resultat $0 \quad , \quad 1 \quad , \quad 9$ spricht für die Anordnung
(24) und wandelt man jetzt zurück, so erhält man die beiden Hilfs-
funktionen 1. Art

$$F_{16}^{19} \quad \text{und} \quad G_{16}^{19} \quad . \quad (25)$$

Dieses Verfahren kann für jede weitere nicht volle Hilfsfunktion des Addierers durchgeführt werden, bis die Begrenzung der maximalen FAN-IN-Zahl die Wahl des Hilfsfunktionenpaares so eindeutig bestimmt, wie in Abb. 3 das Paar mit den Indizes $\begin{smallmatrix} 17 \\ 15 \end{smallmatrix}$ bestimmt ist, das weder nach links noch nach rechts verschoben werden darf.

Berechnung des Elektronikaufwands

Ist der Aufbau eines Addierers festgelegt, wie in den ersten Abschnitten für die vollen Hilfsfunktionen beschrieben und wie im letzten Abschnitt für eine Reihe zusätzlicher nicht voller Hilfsfunktionen beschrieben, so liegt für jede Carry-Funktion und jedes Hilfsfunktionenpaar die Zahl der Eingänge fest. Diese Eingänge sind entweder nur weitere Funktionenpaare oder, wie im Fall der Carry-Funktionen, mindestens ein weiteres Funktionenpaar und genau ein weiteres Carry. Es läßt sich also abzählen, wieviele $C(2)$, $C(3)$, usw. auftreten, das Gleiche gilt für die $FG(1)$, und aus Tabelle II kann dann die Zahl der benötigten OR-, AND-Glieder und Dioden oder Drähte für die Carry-Bildung bestimmt werden. Geschieht dies für einen 25-Bit-Addierer, dann kann mit (5''') der Gesamtaufwand für den Addierer bestimmt werden. Auf diese Weise sind die Zahlen für (13) und (14) bestimmt worden.

Am Beispiel des Addierers in Abb. 4 soll diese Methode demonstriert werden. Der Abb. 4 ist zu entnehmen, daß die Carry-Indizes 1 bis 25 (C_{26} war in Abb. 4 nur als Beispiel für eine volle Stufe mit Funktionen 2. Art angegeben worden) folgendermaßen den $C(2)$ und $C(3)$ zugeordnet werden können:

$C(2)$: 1, 3, 5, 9, 11, 12, 15, 17

$C(3)$: 2, 4, 6, 7, 8, 10, 13, 14, 16, 18, 19, 20, 21, 22, 23, 24, 25,

Das ergibt 8 Funktionen vom Typ $C(2)$ und 17 Funktionen vom Typ $C(3)$. Für die $FG(2)$ und $FG(3)$ gilt die folgende Aufstellung, wobei aber noch bemerkt werden soll, daß die Indizes-Differenz nur bei den Funktionen 1. Art, schlecht aber bei denen 2. Art zur Einordnung benutzt werden kann:

$FG(2)$:
7 12 19 21 23 25
6' 9' 18' 18' 18' 24

$FG(3)$:
5 8 11 14 15 17 17 20 22 23 24 25
3' 6' 9' 12' 9' 15' 9' 18' 18' 21' 18' 18'

Das sind 6 Funktionenpaare vom Typ $FG(2)$ und 12 Funktionenpaare vom Typ $FG(3)$.

Aus Tabelle II folgt dann

$$\begin{array}{lcl} C(2): (1, 1, 4) \times 8 & = & 8, 8, 32 \\ C(3): (1, 2, 8) \times 17 & = & 17, 34, 136 \\ FG(2): (1, 2, 6) \times 6 & = & 6, 12, 36 \\ FG(3): (1, 3, 11) \times 12 & = & 12, 36, 132 \\ & & \hline & & 43, 90, 336 \end{array}$$

Mit (5''') ergibt sich dann für den Gesamtaufwand für 25 Bit

$$93, 190, 686 \quad 5\tau \quad n = 2 \quad (26)$$

Mit diesem Abschnitt sind die Lücken geschlossen, die in der im 1. Teil dieser Arbeit entwickelten Systematik des Addierer-Aufbaues geblieben waren. Ein Addierer, der mit den bisher angegebenen Verfahren aufgebaut ist, soll ein systematischer Addierer im Gegensatz zu Addierern, die noch mehr Elektronik-Einsparungen besitzen, genannt werden.

Die Addierer-Spitze

Die Spitze eines Addierers der Länge m sei die Stelle, an der C_m gebildet wird. Fällt dieses C_m mit dem letzten linken Bit einer Stufe der Carry-Funktionen zusammen, dann ist dieser folgende Abschnitt ohne Bedeutung. Das gleiche gilt, wenn C_m das letzte linke Bit einer Gruppe von Carry-Funktionen im Sinne der Abbildung 5 ist. In allen anderen Fällen ist es sinnvoll, zu prüfen, ob nicht auch an den letzten vollen Hilfsfunktionen Änderungen zum Zwecke der Elektronik-Ersparnis vorgenommen werden sollten. Im Prinzip könnte, hätte man diese Absicht nicht, der systematisch aufgebaute Addierer nach C_m einfach abgebrochen werden.

Ein Beispiel sei wieder der vorhin erwähnte Addierer mit $n = 7$, der jetzt aus 25 Bit bestehen soll. Der systematische Aufbau ergibt für die Spitze, wohl jetzt verständlich und unter Gebrauch der in (25) angegebenen Hilfsfunktionen

$$\begin{array}{ccccccc} & & FG(8)_{16}^{23} & & & & FG(4)_{16}^{19} \\ C(5)_{25} & C(4)_{24} & C(3)_{23} & C(6)_{22} & C(5)_{21} & C(4)_{20} & C(3)_{19} \end{array}$$

Eine Verschiebung von $FG(8)$ in $FG(7)$ änderte nur die Reihenfolge in der Reihe der $C(i)$, wäre also erfolgreich. Die Aufwandsänderung wäre mit Tabelle II 0, -1, -10. Wandelt man aber $FG(8)_{16}^{23}$ in $FG(4)_{20}^{23}$ um und benutzt $FG(4)_{16}^{19}$ wie ein volles Hilfsfunktionenpaar, dann entsteht

$$\begin{array}{ccccccc} & & FG(4)_{20}^{23} & & & & FG(4) \\ C(6) & C(5) & C(4) & C(6) & C(5) & C(4) & C(3) \end{array} \quad (27)$$

und Tabelle II ergibt für

$$FG(8) \rightarrow FG(4) \quad 0, \quad -4, \quad -34$$

$$C(3) \rightarrow C(6) \quad \underline{0, \quad -3, \quad 18}$$

und als Resultat $0, \quad -1, \quad -16$, d.h. (27) ist die bessere Lösung für den 25-Bit-Addierer mit $n = 7$.

Die 1. Stufe

Werden die Carry in mindestens 2 Stufen gebildet, so sieht der systematische Aufbau vor, daß die C_1 bis C_n gleichzeitig gebildet werden. Notwendigerweise muß aber nur C_n , das für die Carry-Funktionen der 2. Stufe benötigt wird, sofort gebildet werden. Die anderen C_i mit $i < n$ müssen nur spätestens mit C_m gebildet werden. Im letzten Beispiel mit $n = 7$ und 2 Stufen für die Carry-Bildung bedeutet dies, daß ursprünglich die C_1 bis C_6 die folgende Konfiguration bildeten:

$$C(7)_6 \quad C(6) \quad C(5) \quad C(4) \quad C(3) \quad C(2)_1. \quad (28)$$

Hieraus kann sofort

$$\begin{array}{cccc} & C(6) & & C(3) \quad C(2) \\ C(2)_6 & & C(3) & C(2) \end{array}$$

und daraus

$$\begin{array}{cccc} & & C(5) & C(3) \quad C(2) \\ C(3)_6 & C(2) & & C(2) \end{array} \quad (29)$$

gebildet werden. Bei jedem Übergang wurden $C(i)$ mit höherem Argument durch solche mit kleinerem Argument ersetzt.

Man kann durch weitere Variationen unter Gebrauch der Tabelle II nachprüfen, daß (29) das günstigste Ergebnis bietet.

Fragt man nach dem Gewinn an Elektronik, so ergibt Tabelle II

für (28): 6, 21, 104, während (29)
 6, 11, 47 ergibt.

Diese Überlegung für die 1. Stufe gilt natürlich bei mehr als 2 Stufen für die Carry-Bildung auch für die 2. Stufe bis zur vorletzten Stufe. Praktisch liegt aber eine Begrenzung vor. Nur für großes n , $n > 3$, läßt sich überzeugend wie oben Elektronik ein-

sparen, je größer n , desto mehr. Je größer n aber ist, desto weniger Stufen sind notwendig, so daß oft nur in der 1. Stufe dieses Verfahren benutzt werden kann.

Zusätzliche Hilfsfunktionspaare

Es sei wieder der Addierer mit 25 Bit und $n = 7$ betrachtet. Die Carry-Funktionen C_8 bis C_{14} könnten ohne Hilfsfunktionen 1. Art aufgebaut werden. Die ursprüngliche Anordnung wäre

$$C(8)_{14} \quad C(7) \quad C(6) \quad C(5) \quad C(4) \quad C(3) \quad C(2)_8. \quad (30)$$

Würde man stattdessen bilden

$$C(7)_{14} \quad C(6) \quad C(5) \quad C(4) \quad C(3) \quad C(2) \quad C(2), \quad FG(2)$$

so ergäben sich die Änderungen

$$\begin{aligned} FG(2) &: 1, 2, 6 \\ C(8) \rightarrow C(2) &: 0, -6, -39 \end{aligned}$$

also einwandfrei weniger Elektronik, wenn ein zusätzliches OR-Glied zugelassen ist.

Aus

$$C(6)_{14} \quad C(5) \quad C(4) \quad C(3) \quad C(2) \quad C(3) \quad C(2), \quad FG(3)$$

folgt dann mit

$$\begin{aligned} FG(2) \rightarrow FG(3) &: 0, 1, 5 \\ C(7) \rightarrow C(3) &: 0, -4, -26 \quad \text{eine weitere Verbesserung.} \end{aligned}$$

Aus

$$C(5) \quad C(4) \quad C(3) \quad C(2) \quad C(4) \quad C(3) \quad C(2), \quad FG(4) \quad (31)$$

folgt mit

$$FG(3) \rightarrow FG(4) : 0, 1, 6$$

$$C(6) \rightarrow C(4) : 0, -2, -13$$

immer noch eine Verbesserung.

Mehr ist aber nicht mit 1 Hilfsfunktionenpaar zu erreichen, eine weitere Verschiebung bringt keine Änderung in den $C(1)$, aber einen Elektronikzuwachs in $FG(5)$. Vergleicht man den Gesamtaufwand für (30), so ergibt Tabelle II

	7, 28, 147,	während
für (31) sich	8, 20, 86	ergibt.

Ineinanderschieben von Stufen

Der in Abbildung 3 gezeigte 25-Bit-Addierer stellt einen Sonderfall dar. Er besteht aus 3 Stufen, die 3. Stufe besteht aber nur aus C_{25} . Würde man C_{24} nicht bilden, wie in Abb. 3 gezeigt, sondern von C_{23} aus in der 3. Stufe C_{24} und C_{25} bilden, dann würde das Hilfsfunktionenpaar 1. Art $\begin{smallmatrix} 24 \\ 19 \end{smallmatrix}$ nicht benötigt, außerdem fände ein Übergang von $C(5)$ nach $C(3)$ statt.

Auf die gleiche Weise könnte man den Beginn der 3. Stufe noch weiter nach rechts verlegen und würde ohne Verlust an Rechenzeit einen Gewinn an Elektronik erzielen. Dieses Verfahren wäre auch dann anwendbar, wenn die 3. Stufe ursprünglich aus mehr als einer Carry-Funktion bestehen würde.

Die Ursache für diese Möglichkeiten liegt in der Tatsache, daß innerhalb jeder Stufe der Elektronikaufwand pro Bit von rechts nach links, oder mit wachsendem Carry-Index, zunimmt. Es kann also besser sein, von der 2. und 3. Stufe nur die rechts gelegenen Teile dieser Stufe zu verwenden. Ein naheliegender Gedanke wäre, dann hier in Abb. 3 die C_5 bis C_{25} gleichmäßig auf die 2 Stufen zu verteilen, so daß z.B. in der 2. Stufe C_5 bis C_{15} , in der 3. Stufe C_{16} bis C_{25} gebildet werden. Dieser Überlegung steht entgegen, daß nur in der 2. Stufe, wie im vorletzten Abschnitt beschrieben, Elektronik dadurch eingespart werden kann, daß eine Reihe dieser Carry der 2. Stufe erst gleichzeitig mit den Carry der 3. Stufe gebildet werden. Diese Überlegung gibt der restlichen 2. Stufe eine größere Länge als der restlichen 3. Stufe. Bei größeren Addiererbitzahlen m würde man zur Lösung des Problems, die richtige Nahtstelle zwischen 2. und 3. Stufe zu finden, einen Digitalrechner einsetzen.

Die Abbildung 6 wurde ohne Digitalrechner aus der Abbildung 3 entwickelt. Es wurde versucht die Zahl der Hilfsfunktionen 1. Art möglichst niedrig zu halten und die Zahl der Eingänge in die Carry-Funktionen möglichst klein zu halten. Auf diese Art von Grundregeln kommt man automatisch, wenn man durch Probieren versucht, das Problem zu lösen.

Der Beginn der 3. Stufe wurde in Abbildung 6 bei C_{20} gewählt. Würde man wieder, wie in vorher besprochenen Fällen, eine Änderung durchführen, z.B. den Anfang der 3. Stufe nach C_{19} verlagern, so ergäbe eine Nachrechnung, daß einem Übergang von $FG(5)$ in $FG(4)$ ein weiterer Übergang von $C(3)_{18}$ in $C(5)_{25}$ gegenübersteht. Diese Änderung ist nachteilig für den Elektronikaufwand. Würde die 3. Stufe bei C_{21} beginnen, so träte ebenfalls ein geringer Mehraufwand von 2 Dioden oder Drähten ein. Auf ähnliche Weise läßt sich die Wahl jeder Funktion in Abbildung 6 begründen.

Zum Abschluß wird noch der Elektronik-Aufwand für beide Addierer der Abbildung 3 und der Abbildung 6 bestimmt. In der unten stehenden Aufstellung sind hinter jedem $C(i)$ und $FG(i)$ die zugehörigen Indexzahlen der Abbildung 3 und 6 aufgeschrieben. Weiter rechts steht die Anzahl der angeführten Indexzahlen und daneben das entsprechende Zahlentripel, das Tabelle II liefert.

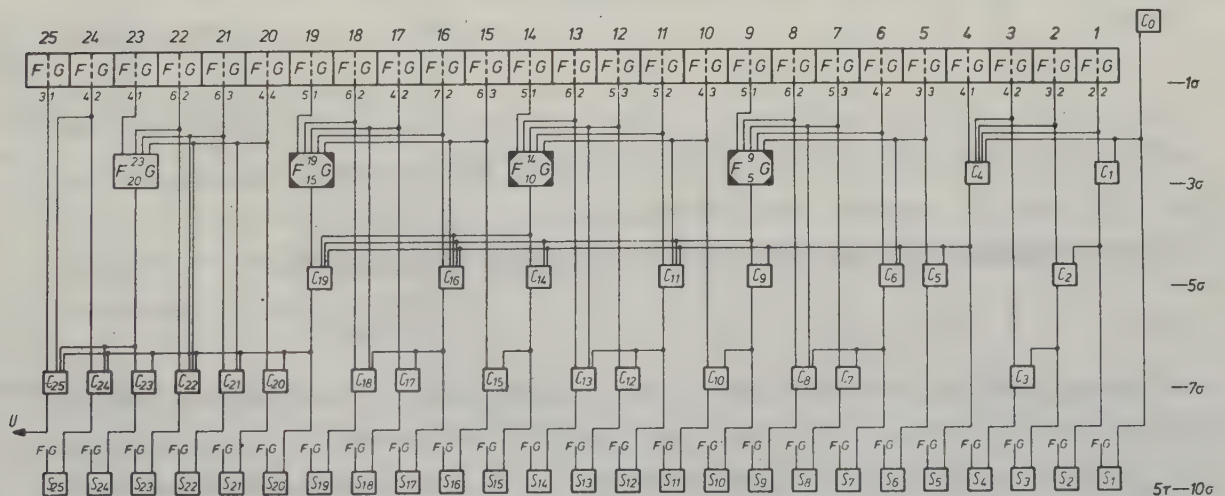


Abb. 6, Addierer, gerast mit Hilfsfunktionen, $n=4$

Das vorweggenommene Ergebnis für den Addierer der Abbildung 6 ist

$$79, \quad 165, \quad 633 \quad 5\tau \quad n = 4 \quad . \quad (32)$$

Dieses Ergebnis ist günstiger als der Aufwand der Gl.(13), die die Zahlen für Abbildung 3 liefert, und sogar günstiger als der Aufwand der Gl.(9) für den 5 x 5 Stufen-Addierer mit der Gesamt-rechenzeit 71.

Elektronikberechnung des Addierers nach Abbildung 6

C(2) :	1, 2, 3, 5, 7, 9, 10,				
	12, 15, 17, 20, 23,	:	12 ergibt	12,	12, 48
C(3) :	6, 8, 13, 14, 18, 21, 24	:	7 "	7,	14, 56
C(4) :	11, 19, 22, 25	:	4 "	4,	12, 52
C(5) :	4, 16	:	2 "	2,	8, 38
FG(4) :	23	:	1 "	1,	4, 17
	20				
FG(5) :	9 14 19	:	3 "	3,	15, 72
	5' 10' 15				
				29,	65, 283
		(5''')		50,	100, 350
Mit (5''')	ergibt sich als Gesamtaufwand:			79,	165, 633

Elektronikberechnung des Addierers nach Abbildung 3

C(2)	:	1, 5, 9, 25	:	4 ergibt	4,	4,	16
C(3)	:	2, 6, 10, 12, 14	:	5 "	5,	10,	40
C(4)	:	3, 7, 11, 13, 15, 17, 19	:	7 "	7,	21,	91
C(5)	:	4, 8, 16, 18, 20, 21, 22, 23, 24	:	9 "	9,	36,	171
FG(2)	:	21 20	:	1 "	1,	2,	6
FG(3)	:	12 17 22 10' 15' 20	:	3 "	3,	9,	33
FG(4)	:	23 20	:	1 "	1,	4,	17
FG(5)	:	9 14 19 24 5' 10' 15' 20	:	4 "	4,	20,	96
							34, 106, 470
				(5''')			
							50, 100, 350
Gesamtaufwand wie in (13) angegeben:							84, 206, 820

Der Carry Look-ahead Addierer nach MacSorley

Der in [2] beschriebene Addierer, der das Carry Look-ahead-Prinzip benutzt, besteht aus gleichen Grundeinheiten, deren Aufbau in Abb. 7 dargestellt ist. Diese Einheit nimmt 5 Paare von F_i -, G_i -Funktionen (von oben in Abb. 7) und ein Carry C_0 (von rechts in Abb. 7) als Eingangswerte an und liefert als Ergebnis einmal 5 (nach oben laufende) Carry und außerdem (nach links in Abb. 7 gehend) entweder ein weiteres Carry oder ein Paar von Funktionen F_s^r und G_s^r . Der Zusammenhang zwischen den eingehenden Werten und den nach oben laufenden Carry ist der gleiche wie im unteren Drittel der Abb. 1. Verbindet man wie dort die an der oberen Kante der Abb. 7 befindlichen Ein- und Ausgänge mit den gleichbenannten Ein- und Ausgängen der Zeichnung im oberen Drittel der Abb. 1, so erhält man einen Addierer mit der Bit-Zahl 5 und der Gesamtrechnzeit von 6σ oder 3τ wie bei jedem einstufigen Addierer.

Es ist der Abb. 7 zu entnehmen, daß von jedem Eingang her zu jedem Ausgang hin höchstens 2 logische Glieder durchlaufen werden, so daß man sagen kann, der Signaldurchlauf durch eine solche Grundeinheit benötigt die Zeit 2σ .

In der linken oberen Ecke der Abb. 7 befinden sich 2 AND-Glieder, deren Kästen gestrichelt umrandet sind. Es wird immer nur eines dieser Glieder benutzt, in der oberen Lage werden F_s^r - und G_s^r -Funktionen erzeugt, in der tieferen Lage wird das C_5 erzeugt, der F_1^5 -Ausgang ist dann leer.

Diese Grundeinheit wird auch zur Bildung von Hilfsfunktionen benutzt und für diesen Fall sind die Ausgänge F_s^r und G_s^r strichliniert an die Unterkante der Abb. 7 hinuntergeleitet worden, wo sie mit dem strichlinierten Eingang C_0 genauso ein Ein- und Ausgangstrippel bilden, wie es an der Oberkante der Abb. 7 fünfmal zu finden ist. Der Verfasser sieht dann in diesen F_s^r - und G_s^r -Funktionen diejenigen F- und G-Funktionen, die bei der Addition von zwei 32-wertigen Ziffern entstehen, im Gegensatz zu den oberen F_1, G_1 Funktionen, die aus der Addition von zwei 2-wertigen Ziffern entstehen. Setzt man mehrere dieser Grundeinheiten vertikal übereinander, dann können die untersten F, G-Funktionen als bei der Addition zweier noch höherwertiger Ziffern entstanden gedacht werden.

Das C_0 , das von unten in die Abb. 7 hineinführt, ist bei dieser Betrachtungsweise im einfachsten Falle das Carry für eine 32-wertige Ziffer, das nur noch zur Bildung eines dieser 32 Werte innerhalb dieser Grundeinheit beiträgt.

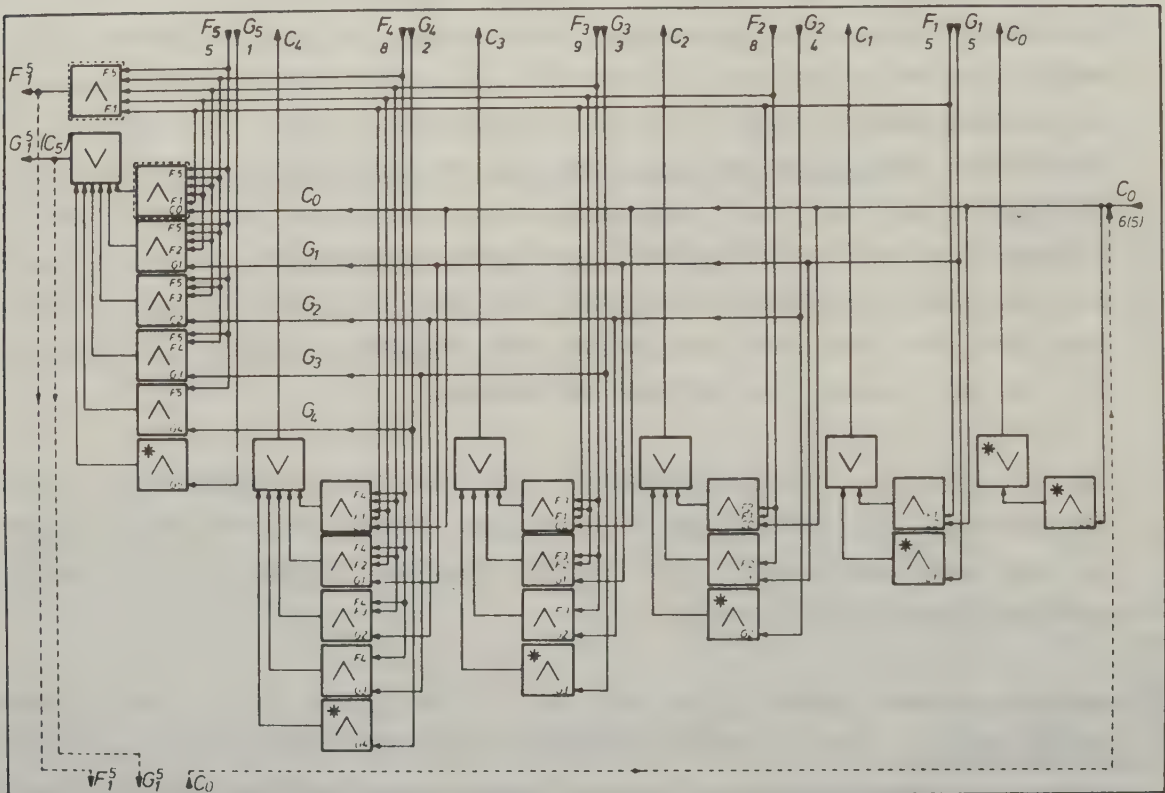


Abb. 7 Grundeinheit nach MacSorley

Für den später durchgeführten Vergleich des Elektronikaufwands sind in Abb. 7 noch 7 logische Glieder mit einem * versehen, diese Glieder könnten auch fehlen, jedes hat nur je einen Eingang und je einen Ausgang.

Für den Aufbau eines Addierers aus diesen Grundeinheiten sind nur noch die 5 Tripel von Ein- und Ausgängen am oberen Rand von Abb. 7, sowie entweder das Tripel an der unteren Kante oder das aus dem seitlichen Eingang und Ausgang gebildete Paar von Interesse. Man kann wie in den Abb. 8 eine solche Einheit durch einen Kasten darstellen, an dem nur die benutzten Ein- und Ausgänge gezeichnet sind.

In der Abb. 8a ist ein 20-Bit-Addierer gezeigt, die Kästen für die Bildung der F_i , G_i und S_i sind nicht aufgeführt. Der Zeichnung kann entnommen werden, daß das von C_0 ausgehende Signal durch alle diese 4 Kästen laufen muß, um C_{20} erzeugen zu können, d.h. es wird hierfür die Zeit 8σ benötigt. Dazu kommen 4σ für die Bildung der F_i , G_i und S_i , wie in Abb. 1, so daß der Addierer eine Gesamtrechnzeit von 12σ oder 6τ hätte. Wäre noch ein 5. Kasten für insgesamt 25 Bit angeschlossen, käme man mit 7τ auf genau die Zeit, die der in Abb. 2 angegebene Addierer benötigt. Der Aufbau der beiden Addierer wäre dann auch bis auf die in Abb. 7 mit einem * versehenen logischen Glieder identisch.

Bei der Besprechung der Abb. 2 war nicht von Hilfsfunktionen die Rede, diese tauchten erst später (Abb. 3 und 4) auf. MacSorley benutzt aber eine andere Nomenklatur, er sagt, dieser Addierer sei aus Carry-Look-ahead-Gruppen vom Level I gebildet, der langsamste Addierer, der gemäß den Gl.(3) gebildet worden wäre, hätte den Level Null. Dementsprechend ist in die Kästen der Abb. 8a eine I eingezeichnet worden.

Der Gebrauch von Level II Logik geht nach MacSorley dann so vor: In den Kästen der Abb. 8a werde nicht das nach links gehende Carry, sondern durch Gebrauch des anderen umrandeten AND-Gliedes in Abb. 7 wurden die Funktionenpaare F_S^r und G_S^r gebildet und nach unten in Abb. 8b herausgeführt. Diese Ausgänge werden in ein weiteres Grundelement, durch II gekennzeichnet, hineingeführt und die Level I-Kästen erhalten von dort das jeweilig benötigte Carry. Dieses neue Grundelement dient als Carry-Look-ahead-Logik von Level II. Der weiteste Signalweg ist der von F_1 nach C_{19} und die dafür benötigte Zeit beträgt nur noch 6σ , die Gesamtrechnzeit also 5τ . Das C_0 wird rechts in den Kasten eingeleitet.

Wäre wieder ein 5. Kasten für einen 25-Bit-Addierer angeschlossen, so hätte dafür das II. Element noch ein Ein- und Ausgangstripel frei, die Gesamtrechnzeit bliebe bei 5τ . Die Gesamtrechnzeit eines Addierers mit Hilfsfunktionen 1. Art, wie im ersten Teil dieser Arbeit angegeben, wäre aber 4τ , wenn $n = 5$ zugrunde liegt.

Diese Annahme ist aber berechtigt, da zur Bildung von C_{25} in Abb. 8b ein maximales FAN-IN von 6 notwendig ist. Es ist interessant festzustellen, daß in Abb. 8b der Gebrauch der Level II Grundeinheit erst bei einem Addierer von mehr als 15 Bit nützlich ist. Das Carry C_{14} z.B. kann in der Abb. 8a und 8b nur gleichschnell gebildet werden.

Der weitere Aufbau von Addierern geht so analog vor, wie aus einem Kasten I die Abb. 8a entstand, und wo es sich zeigte, daß bei 4 Kästen in Abb. 8a eine Beschleunigung durch die Anordnung in Abb. 8b entstand. Besitzt der Addierer bis zu 50 Bit, so wird eine 2. Grundeinheit als Level II-Einheit links an die erste angeschlossen. Übergeben wird C_{25} . Die Gesamtrechnzeit steigt um 1 auf 61, durch ein weiteres Level-II-Element auf 71 bei bis zu 75 Bit. Hat der Addierer bis zu 100 Bit, so kann eine Level-III-Einheit die 4 Level-II-Einheiten so verbinden, wie es in Abb. 8b die eine Level-II-Einheit mit den Level-I-Einheiten getan hat. C_0 würde dann rechts in die Level-III-Einheit eintreten und diese würde C_0 , C_{25} , C_{50} , C_{75} an die Level-II-Einheiten abgeben. Die Rechenzeit bliebe dabei bei 71 und in dieser Zeit könnten auch für 125 Bit die Carry gebildet werden, da die Level-III-Einheit 5 Level-II-Einheiten mit den notwendigen Carry versorgen könnte.

Betrachtet man die Einführung von Level-III-Elementen als äquivalent mit der Einführung von Hilfsfunktionen 2. Art, so kann für $n = 5$ der Abb. 5 entnommen werden, daß der Addierer 215 Bit lang sein dürfte und daß die Gesamtrechnzeit 51 betragen würde. Bei jedem Level nach MacSorley scheint die Rechenzeit aber um 21 zuzunehmen und man möchte verstehen, wie dies zustande kommt. In der Bitzahl sind dagegen die Addierer nicht sehr verschieden, wie nachher noch gezeigt wird.

Versucht man ein paar Elemente der Abb. 8b im Sinne von Abb. 3 umzuzeichnen, wobei vor allem eine von oben nach unten laufende Zeitachse eingeführt wird, so müssen die Level-I-Kästen in 2 Zeitebenen gezeichnet werden, sie werden in der 2. Ebene mit I' bezeichnet, es sind natürlich keine neuen Grundelemente. In der obersten Reihe der Abb. 8c werden in den gezeichneten Level-I-Kästen nur die in den ersten 20 gebildeten Funktionen F, G_6^{10} eingezeichnet.

In den nächsten 2σ wird in den Level-II-Kästen mit Hilfe der Funktionen F, G_1^5 und von C_0 das Carry C_{10} gebildet. In der 3. Ebene, in den dritten 2σ , werden dann im Kasten I' von C_{10} und den entsprechenden F_i, G_i die C_{11} bis C_{14} gebildet. In Abb. 3 wurden C_{11} bis C_{14} gleichzeitig mit C_{10} gebildet. Wollte man dies in Abb. 8c auch versuchen, dann müßte z.B. in C_{14} neben den F_i, G_i mit $i = 11, 12, 13, 14$ noch F, G_6^{10} , F, G_1^5 und C_0 eingehen, die maximale FAN-IN-Zahl wäre zu hoch. Man benötigte weitere F, G_s^r -Funktionen, um die FAN-IN-Zahl zu senken. Diese F, G_s^r -Funktionen müssten im oberen Level-I-Kasten der Abb. 8c gebildet werden, d.h. sie würden den Rahmen dieser Grundeinheiten sprengen.

Man sieht erst durch diesen Vergleich, daß die Geschwindigkeit des Addierers in Abb. 3 genau durch diese nicht vollen Hilfsfunktionenpaare erreicht wurde, die zunächst nicht so genau festlegbar waren, deren Existenz aber dafür sorgte, daß ohne Überschreitung der maximalen FAN-IN-Zahl jedes Carry in der 2. Stufe gebildet werden konnte.

Es soll hier noch auf einen Vorteil hingewiesen werden, den diese Grundeinheiten von Mac-Sorley besitzen. In Abb. 7 wird an den Eingängen angegeben, wie hoch das FAN-OUT ist, mit dem dieser Eingang die Signalquelle belastet. Das FAN-IN ist 5 oder 6, das FAN-OUT in einer Gruppe nie doppelt so groß. In Abb. 3 werden weit größere FAN-OUT verlangt. Auf der anderen Seite kann dies nicht als ernstes Problem von MacSorley betrachtet worden sein, denn in seiner Arbeit hatte jedes Logik-Element sowohl einen positiven wie einen negativen Ausgang, d.h. die Zeit für diese Bildung beider Werte nennt er σ , und dies gestattet in der Regel, wie im letzten Abschnitt ausgeführt, ein hinreichendes FAN-OUT durch parallele Negatoren.

Die Bitzahl, die mit Level-II- oder Level-III-Elementen erreichbar ist, ist größer als sie MacSorley angibt. In der Zeit, in der in Abb. 8b F, G_6^{10} gebildet werden, könnten auch im rechten Level-I-Kasten mit Hilfe von einem direkt eingeführten C_0 die Carry C_1 bis C_5 gebildet werden. Dann würde man C_5 statt C_0 von rechts in den Level-II-Kasten einführen und könnte mit diesem die Carry von

C_5 bis C_{30} statt von C_0 bis C_{25} bilden. Endlich würde im Level-II-Kasten C_{30} erzeugt und dazu verwendet werden können, um in einem 7. Level-I-Kasten die Carry C_{30} bis C_{35} bilden zu können. Der Weg von F_1 zu C_{34} führt durch 3 Kästen, das Signal benötigt die Zeit von 6σ , die Gesamtrechnzeit bleibt 5τ . Abb. 8d gibt eine Skizze dieses Addierers wieder. Ein Addierer nach Abb. 3 mit $n = 5$ kann aber auch nur 35 Bit bilden, allerdings in 4τ . Benutzt man ein Level-III-Element, so kann analog in dieses Element von rechts statt C_0 ein C_{30} eingeführt werden, links gibt dieses Element ein C_{155} ab, das letzte Carry ist dann C_{185} . C_{215} war der vorhin erwähnte Wert für $n = 5$ und Hilfsfunktionen 2. Art.

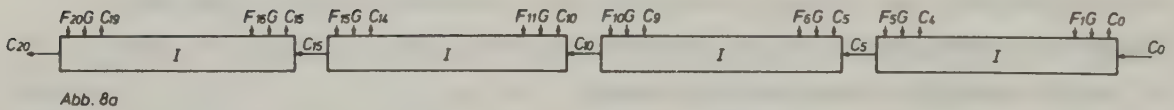


Abb. 8a

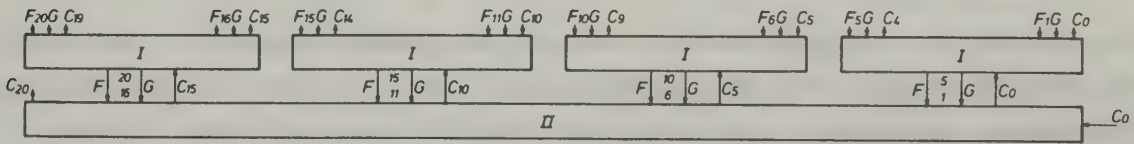


Abb. 8b

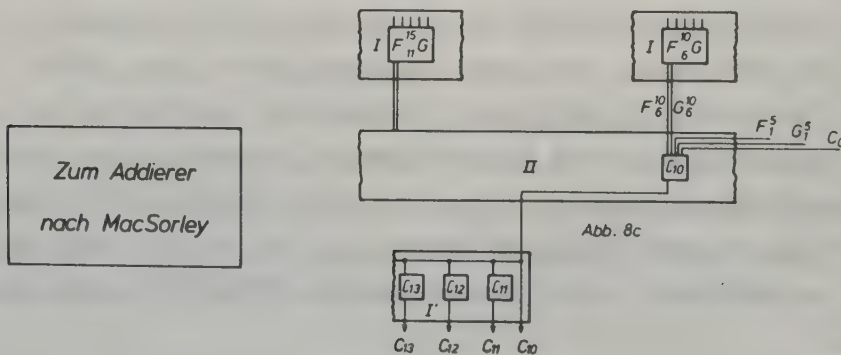


Abb. 8c

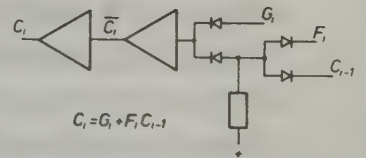


Abb. 8e

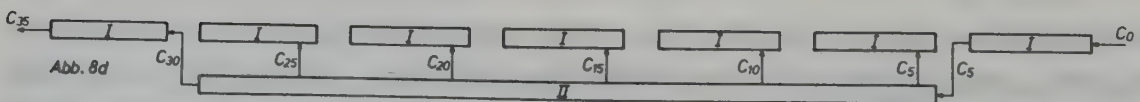


Abb. 8d

Der Look-ahead-Addierer nach Flores

MacSorley, den Flores anerkennend zitiert, führt in [2] aus, daß bei einem einfachen Addierer (z.B. gemäß (3)) die Zahl der Bit, über die ein Carry fortgepflanzt wird, im Durchschnitt gleich $\log_2(m)$ ist. m ist die Zahl der Bit im Addierer. Für einen 50-Bit-Addierer gibt MacSorley eine Durchschnitts-Additionszeit von 5,6τ an, während er für den von ihm beschriebenen Look-ahead-Addierer 6τ angibt. Da außerdem der Look-ahead-Addierer nach seinen Angaben einen Elektronik-Mehraufwand von 60% besitzt, legt MacSorley in [2] nach einer genauen Beschreibung des Look-ahead-Addierers dar, wie der - in der Literatur "Asynchron-Addierer" genannte - Addierer möglichst Elektronik sparend aufgebaut werden kann. Dieser Asynchron-Addierer stellt selber den Zeitpunkt fest, an dem der Additionsvorgang beendet ist, gibt dem Rechner ein entsprechendes Signal und ist dann im Mittel schneller als der von MacSorley beschriebene Look-ahead-Addierer.

Man kann nur vermuten, daß aus diesem Grund der Look-ahead-Addierer wenig weitere Beachtung fand, oder, daß er, falls er verwendet wurde, nur in der Stufenform der Abb. 2, die leicht zu übersehen ist, gebaut wurde, d.h. ohne Hilfsfunktionen. (Siehe dazu auch [5], Seite 256 und 285). Nur unter diesen angenommenen Voraussetzungen läßt sich verstehen, wie Flores in [3] den Look-ahead-Addierer beschreibt.

Zunächst geht Flores, wie die meisten Autoren, auf den Stufen-Addierer ein (hier in Abb. 2) und gibt genau an, in welchen Zeitintervallen die Carry der einzelnen Stufen gebildet werden. Dann wendet er sich dem Addierer mit Hilfsfunktionen zu und er knüpft seine Überlegungen an eine Gl. wie (3'') an, die man in der Form

$$C_i = \bigwedge_{j=0}^i \prod_{k=j+1}^i F_k G_j \quad (3''')$$

schreiben kann. Dabei wird C_0 in G_0 umbenannt und für das Produkt $\prod_{i+1}^i$ wird immer der Faktor 1 genommen.

Formal werden dann Hilfsfunktionen 1. Art dadurch gebildet, daß in (3''') F- und G-Funktionen wie in (10) ausgeklammert werden. Es werden dazu der Reihe nach, bis zur Spitze m hin, die Paare

$$F_1^5, G_1^5, F_6^{10}, G_6^{10}, F_{11}^{15}, G_{11}^{15} \text{ usw. gebildet. Flores schreibt}$$

dafür $F_1^I, G_1^I, F_2^I, G_2^I, F_3^I, G_3^I$ usw.. Die Verwendung des Buchsta-

ben P statt F soll hier kurz notiert werden.

Zur Bildung von Hilfsfunktionen 2. Art werden nur Funktionen 1. Art zusammengefaßt, es werden gebildet

$$F_1^{25}, G_1^{25}, F_{26}^{50}, G_{26}^{50} \text{ usw., die Indizierung bei Flores lautet}$$

$$F_1^{II}, G_1^{II}, F_2^{II}, G_2^{II} \text{ usw..}$$

Sind die Hilfsfunktionen gebildet, dann werden in (3''') möglichst viele F_i und G_i durch Hilfsfunktionen 1. Art ersetzt. Sollen Hilfsfunktionen 2. Art benutzt werden, werden möglichst viele Paare Hilfsfunktionen 1. Art durch solche von 2. Art ersetzt. Man darf sich vorstellen, daß dies für jedes C_i durchgeführt wird. Der Nachteil des Verfahrens liegt darin, daß jedes C_i direkt auf C_0 , das bei Flores G_0 heißt, zurückgeführt wird, ohne Berücksichtigung der Stufe, in der das C_i gebildet wird.

Flores gibt dann einen Addierer mit Hilfsfunktionen 1. Art an, die Zeichnung ähnelt der Abb. 2 dieser Arbeit, wenn man sich dort über den Carry-Kästen der 2. bis 5. Stufe je einen Kasten zur Bildung eines Hilfsfunktionenpaares 1. Art gezeichnet denkt. Dieser Kasten sollte in der Höhe der vorangehenden Carry-Stufe liegen und die Funktionen aus den F_i und G_i der vorangehenden Stufe bilden. Flores gibt dann noch ([3], Seite 92) genau an, in welchen Zeitintervallen die Carry der einzelnen Stufen gebildet werden, es sind die gleichen Carry in den gleichen Intervallen wie beim einfachen Stufenaddierer, beschrieben in [3], Seite 87. Die Frage erscheint berechtigt, welchen Vorteil bringt der Mehraufwand für die Hilfsfunktionen 1. Art.

Es liegt hier sicher kein Schreibfehler vor, denn auf Seite 92 führt Flores aus, daß Hilfsfunktionen 1. Art "an advantage approximately proportional to the group size" geben, während er für den Gebrauch von Hilfsfunktionen 2. Art in ([3], Seite 93) ausführt: the advantage of this method is proportional to the square of the group size. Die Gruppengröße ist in beiden Fällen 5. Bei der Erläuterung des Gebrauchs von Hilfsfunktionen 2. Art führt Flores sinngemäß weiter aus, daß in der 1. Stufe die Carry C_1 bis C_5 , in der 2. Stufe unter Gebrauch eines Hilfsfunktionenpaares 1. Art die Carry C_6 bis C_{10} und in der dritten Stufe die Carry C_{11} bis C_{35} mit Hilfsfunktionen 2. Art gebildet werden ([3], Seite 93). Über die Indizes dieses Hilfsfunktionenpaares wird nichts gesagt, der Addiereraufbau ist in dieser Art aber schwer zu verstehen, wenn das Hilfsfunktionenpaar die Indizes ²⁵₁ hatte, wie es den Definitionsvorschriften entsprechen würde.

In einer Zusammenfassung über mehrere Addierer-Typen schreibt Flores in dem Absatz, der dem Look-ahead-Addierer gewidmet ist ([3], Seite 114), daß mehrere äquivalente Konfigurationen bei einer vorgegebenen Addierer-Bitzahl m denkbar sind. Er gibt als Beispiel eine Konfiguration für einen 48-Bit-Look-ahead-Addierer an, die ein Paar Hilfsfunktionen 2. Art benutzt. Zu diesem Addierer werden die Zahlen der benötigten Bauteile berechnet. Flores setzt dabei an, daß zu einem OR-Glied bis zu maximal 8 AND-Glieder gehören können. Da aber jedes OR-Glied einen direkten Eingang ohne AND-Glied besitzt, beinhaltet diese Aussage ein maximales FAN-IN von 9. Widersprüchlich ist dann die Angabe, daß einige der AND-Glieder bis zu 8 Eingänge haben können, diese Zahl müßte dann ebenfalls 9 sein. Es ist aber mindestens ein maximales FAN-IN von 8 notwendig, und dann ist die Gruppengröße $n = 5$ zu klein. Mit $n = 7$ könnte der 48-Bit-Addierer ohne Hilfsfunktionen 2. Art in 2 Carry-Stufen mit der Gesamtrechnenzeit von 4τ arbeiten (Abb.5), Flores gibt für seinen Addierer 5τ an.

Der gleiche 48-Bit-Addierer würde bei MacSorley 6τ benötigen, da dort zwei Kästen mit Level-II-Elementen stufenförmig hintereinander geschaltet werden würden. Flores berechnet weiter für den bei MacSorley schnelleren asynchronen Addierer die Zeit 7τ , das

heißt der Look-ahead-Addierer ist bei Flores schneller. Es ist aber nicht auszuschließen, daß sich hier nur die praktisch größere maximale FAN-IN-Zahl auswirkt. Hätte MacSorley ein FAN-IN von 8 benutzt und eine Gruppengröße von 7 gewählt, dann würde der von ihm aufgebaute Look-ahead-Addierer ebenfalls in 5τ das Ergebnis liefern, während der asynchron-Addierer nicht im gleichen Maß durch größeres maximales FAN-IN beschleunigt werden kann. MacSorley hätte dann möglicherweise auch dem Look-ahead-Addierer den Vorzug gegeben.

Da aber, wie oben gezeigt, es prinzipiell möglich ist, den 48-Bit-Look-ahead-Addierer mit $n = 7$ so zu bauen, daß die Gesamt-rechenzeit 4τ ist, und Abb. 5 zeigt, daß dies auch noch mit $n = 6$ möglich ist, darf wohl der Schluß gezogen werden, daß der Look-ahead-Addierer die zur Zeit schnellste Addierer-Form darstellt. Daß der Look-ahead-Addierer die meiste Elektronik verbraucht, soll ebenfalls dazu bemerkt werden.

Dieser Abschnitt soll nicht beschlossen werden, ohne daß die Vorzüge des unter [3] zitierten Buches ausdrücklich hervorgehoben werden. Es scheint zur Zeit kein Buch zu geben, daß in einer solchen breiten Übersicht über die Methoden des Aufbaus des Arithmetik-Teils der Digitalrechner so gründlich referiert und so leicht zu lesen ist. Würde man sich für die Beurteilung dieses Buches auf die Beschreibung des Look-ahead-Addierers stützen, käme man sicher zu einem Fehlurteil.

Der Addierer nach Weinberger und Smith

Über die in [1] zitierte Arbeit soll aus zwei Gründen berichtet werden. Einmal ist es die wohl älteste Veröffentlichung über den Look-ahead-Addierer, die schon einige Punkte vorwegnahm, die offenbar später wieder unbeachtet blieben. Zweitens gibt diese Arbeit ein Beispiel für das Vorgehen, wenn die bisher gemachte Voraussetzung, daß die maximalen FAN-IN-Zahlen der OR-Glieder und der AND-Glieder gleich sind, fortfällt.

Die den Autoren zur Verfügung stehenden logischen Elemente waren OR-Glieder, die 4 Eingänge hatten, jeder Eingang war ein AND-Glied, das wiederum 6 Eingänge hatte, und diese Eingänge dürfen nochmals als OR-Glieder mit je 2 Eingängen ausgebildet sein. Damals wurden noch Elektronenröhren mit entsprechender Dioden-Logik für den Digitalrechnerbau benutzt.

Die Verfasser geben eine Reihe von Verfahren an, die aus dem maximalen FAN-IN von 4 (durch die 4 Eingänge am 1. OR-Glied) praktisch ein maximales FAN-IN von 5 oder von 6 erzeugen. Dabei wird die größere Zahl der Eingänge an den AND-Gliedern und den davorgeschalteten OR-Gliedern und eine entsprechende Umformung der logischen Funktionen benutzt.

Die erste Änderung dieser Art besteht in der Umwandlung von

$$G_1 + F_1 C_0 \quad \text{in} \quad F_1 (A_1 + C_0) (D_1 + C_0). \quad (33)$$

Ist G_1 gegeben, so ist auch F_1 , A_1 und D_1 gegeben, ist F_1 und C_0 gegeben, so sind ebenfalls beide Ausdrücke wahr. In den negativen Werten besteht auch Übereinstimmung. Mit dieser Änderung kann das Carry der 1. Stufe mit dem höchsten Index in der Form

$$C_4 = G_4 + F_4 G_3 + F_4 F_3 G_2 + F_4 F_3 F_2 F_1 (A_1 + C_0) (D_1 + C_0) \quad (33')$$

geschrieben werden, d.h. das maximale FAN-IN ist praktisch auf 5 gestiegen, und mit der Stufenlänge 4 läßt sich ein Stufenaddierer analog Abb.2 aufbauen, wie es die Verfasser auch angeben.

Eine andere benutzte Form mit einem praktisch maximalen FAN-IN von 6 ist

$$C_5 = G_5 + F_5 G_4 + F_5 F_4 (G_3 + F_3) (G_3 + G_2) + \\ + F_5 F_4 F_3 F_2 (G_1 + F_1) (G_1 + C_0). \quad (34)$$

Die Verfasser geben diese Form nicht für C_5 , sondern für Carry mit höherem Index, die mit Hilfsfunktionen 1. Art gebildet werden, an, man überzeugt sich aber der Reihe nach, daß C_5 genau in den Fällen wahr ist, in denen auch die Originalgleichung nach (3'') wahr wäre. Im Prinzip wurde zweimal

$$G_i + F_i G_{i-1} \quad \text{in} \quad (G_i + F_i) (G_i + G_{i-1}) \quad (35)$$

verwandelt. Die Gleichung für C_5 nutzt sowohl die maximale FAN-IN-Zahl des OR-Gliedes und in einem Fall die des AND-Gliedes aus. In diesem letzten Fall liegt die Begrenzung, denn mit der Umwandlung der beiden ersten Terme (34) von

$$G_5 + F_5 G_4 \quad \text{in} \quad F_5 (A_5 + G_4) (D_5 + G_4) \quad \text{nach} \quad (33)$$

$$\text{oder} \quad \text{in} \quad (G_5 + F_5) (G_5 + G_4) \quad \text{nach} \quad (35)$$

$$\text{oder} \quad \text{in} \quad (A_5 + D_5) (A_5 + A_4) (A_5 + D_4) (A_4 + D_5) (D_4 + D_5) \quad ,$$

wie von den Verfassern angegeben, wird zwar ein Eingang am OR-Glied frei, es ist aber nicht möglich, dem letzten Term in (34) noch ein weiteres F_1 , z.B. F_6 , anzuhängen. Das wäre aber notwendig, wollte man die maximale FAN-IN-Zahl auf den Wert 7 erhöhen.

Unter Gebrauch dieser Verfahren geben dann die Verfasser nach der 1. Stufe mit 4 Carry (33') eine 2. Stufe mit 25 Carry unter Gebrauch von Hilfsfunktionen 1. Art an. Die weiteren Stufen haben immer 25 Carry, wenn nur Hilfsfunktionen 1. Art benutzt wurden. Werden für die 3. Stufe Hilfsfunktionen 2. Art benutzt, dann hat die 3. Stufe 150 Carry, der ganze Addierer liefert 179 Bit in der Gesamtrechnenzeit von 5τ . Ein 25-Bit-Addierer würde in 4τ arbeiten.

Diese Zeiten wurden in keiner der zitierten späteren Arbeiten erreicht. Die Verfasser geben als Beispiel einen 53-Bit-Addierer an, bei dem sie schon angeben, daß es sinnvoll sei, die Stufen ineinander zu schieben, und nur die rechten Teile der Stufen zu benutzen, die relativ wenig Elektronik verwenden. Die in dieser Arbeit benutzte Darstellung der Abb. 2, 3, 4 und 6 ist an die Darstellung dieses 53-Bit-Addierers angelehnt.

Methodenvergleich an einem 25-Bit-Addierer

Die Abb. 5 zeigt am deutlichsten, daß der Aufbau eines Addierers durch die maximale Bitzahl m , die hier immer 25 sein soll, und durch die maximale FAN-IN-Zahl $n + 1$ bestimmt ist. Aus diesen Angaben folgt die Gesamtrechnenzeit, in τ gemessen, und die Länge der 1. Stufe gleich n .

Will man Vergleiche durchführen, dann ist es sinnvoll, solche Addierer miteinander zu vergleichen, bei denen auch die FAN-IN-Zahl $n + 1$ übereinstimmt.

Eine zweite Möglichkeit für einen Vergleich besteht darin, daß für eine Reihe von Addierern das maximale FAN-IN geändert wird, aber immer die gleiche Methode für den Aufbau für die Addierer verwendet wird.

Als Vergleichsobjekt bietet sich der Addierer von MacSorley an. Der schnellste Addierer wird aus 6 Grundeinheiten der Abb. 7 in einer Konfiguration nach Abb. 8b aufgebaut. Die Abb. 7 enthält 5 OR-Glieder, 15 AND-Glieder, wobei die Elemente mit * nicht mitgezählt wurden und die gestrichelt umrandeten AND-Glieder nur einmal gezählt wurden. Die Dioden-Zahl ergibt sich durch Abzählen oder Gebrauch der Formeln $(4''')$ und $(4''''')$ zu 70, wenn C_5 gebildet wird. Die Änderungen für den Fall, daß F, G_1^5 gebildet wird, sind 2 Dioden weniger. Es ergeben sich also für alle Grundeinheiten, die links einen Carry-Ausgang bilden, das Tripel

$$C_{51} : 5, 15, 70 \quad \text{größtes FAN-IN} = 6 \quad (36)$$

und für alle Grundelemente, die die F,G-Funktionen bilden

$$F, G_{5(i-1)+1}^{5i} : 5, 15, 68 \quad \text{größtes FAN-IN} = 5. \quad (37)$$

Analog zu Abb. 8b erhält man dann 5 Level-I-Kästen mit dem Tripel (37) und 1 Level-II-Kasten, der das Carry C_{25} bildet und dem das Tripel (36) zugeordnet ist. Der Gesamtaufwand für die Carry-Bildung beträgt daher

$$30, 90, 410, \quad (38)$$

der Gesamtaufwand für den Addierer mit (5''')

$$80, 190, 760 \quad 5\tau \quad n=5 \quad (39)$$

Die Gesamtadditionszeit beträgt 5τ und das benutzte maximale FAN-IN ist 6.

Baut man mit dieser FAN-IN-Zahl einen Addierer nach Abb. 5 auf und wendet alle Möglichkeiten zur Elektronik-Einsparung an, dann entsteht ein Addierer nach Anlage 1, dessen Tripel

$$31, 95, 437 \quad (40)$$

lautet, woraus für den Gesamtaufwand

$$81, 195, 787 \quad 4\tau \quad n=5 \quad (41)$$

wird. Die Gesamtadditionszeit beträgt 4τ , d.h. die schnellere Rechenzeit ist mit knapp 5% Mehraufwand gegenüber (38) erkaufte.

Würde man aber die Rechenzeit von 5τ als genügend betrachten, dann kann der Addierer in 3 Stufen aufgebaut werden. In Anlage 2 ist ein solcher Addierer beschrieben, das Tripel für die Carry-Elektronik lautet jetzt

$$29, 65, 282 \quad (42)$$

d.h. rund 20% weniger als (38) und die Zeile für den Gesamtaufwand lautet

$$79, 165, 632 \quad 5\tau \quad n=5 \quad (43)$$

Man könnte nun gerade bei obigem Addierer (39) darüber streiten, ob ein maximales FAN-IN von 6 oder von 5 vorliegt. Das einzige Carry, das mit einem FAN-IN von 6 gebildet wird, ist C_{25} . Dieses Carry wird aber ein ganzes τ vor den Carry C_{21} bis C_{24} gebildet. Man könnte gleichschnell statt C_{25} die Funktionen F_1^{25} und G_1^{25} bilden, und

dann daraus gleichzeitig mit C_{21} bis C_{24} auch C_{25} bilden. Das würde nur ein weiteres $C(2)$ kosten. Die Elektronik bestände dann aus 6 Kästen nach (37) und dem erwähnten $C(2)$ -Element: Die Summe wäre

$$31, \quad 91, \quad 412 \quad (44)$$

woraus mit (5''') wieder

$$81, \quad 191, \quad 762 \quad 5\tau \quad n=4 \quad (45)$$

entsteht.

Baut man mit dieser FAN-IN-Zahl einen Addierer nach Abb. 5, so besteht er aus 3 Stufen, die 3. Stufe enthält aber nur C_{25} (Abb.3). Man wird statt dessen einen Addierer mit einer Anordnung nach Abb. 6 wählen. Die Elektronikberechnung ist für beide Addierer anschließend an Abb. 6 aufgeführt und es gibt statt (44) das Tripel

$$29, \quad 65, \quad 283 \quad (46)$$

woraus sich mit (5''') wieder

$$79, \quad 165, \quad 633 \quad 5\tau \quad n=4 \quad (47)$$

ergibt.

Der Vergleich zwischen dem Addierer nach Abb. 3 und den Zahlen in (44) und (45) ergibt günstigere Zahlen für den Addierer nach Mac-Sorley, hier macht sich bemerkbar, daß in Abb. 3 die gesamte 2. Stufe mit dem hohen Elektronikaufwand am linken Ende benutzt wird.

Zum Abschluß soll nochmals erwähnt werden, daß der Addierer nach [1] ebenfalls für $m = 25$ in 4τ Gesamtrechnenzeit arbeitet.

Die vorher festgestellten Zahlen sind noch einmal in einer Tabelle zusammengestellt. Dabei enthält die 5. Spalte die Summe der beiden ersten Spalten, außerdem wurden noch (6'), (9) und (6'') mit aufgeführt

75,	425,	3960	3 τ	500	nicht erreichbar	(6')
81,	195,	787	4 τ	276	n=5 Anlage 1	(41)
79,	165,	632	5 τ	244	n=5 Anlage 2	(43)
79,	165,	633	5 τ	244	n=4 Abb. 6	(47)
75,	175,	700	7 τ	250	n=5 5x5 Stufen	(9)
80,	190,	760	5 τ	270	n=5 MacSorley	(39)
75,	125,	450	26 τ	200	langsamster Addierer	(6'')

Die vorstehenden Werte sind nach wachsenden τ geordnet worden. Eine Ausnahme macht (9), das im Elektronikaufwand günstiger ist als (39), auf der anderen Seite aber ungünstiger als (43) und (47) und außerdem noch langsamer als diese. Der in der Literatur angegebene Stufen-Addierer beim Gebrauch von Carry-Look-ahead-Logik ist also nicht nur langsamer, das ist er von der Konstruktion her, sondern auch aufwendiger im Elektronik-Verbrauch als ein schnellerer Addierer, der mit Hilfsfunktionen arbeitet, wenn man wie in (43) oder (47) darauf verzichtet, das Maximum an Geschwindigkeit wie in (41) anzustreben.

In der Anlage 1 und 2 wurden je ein Addierer mit Hilfe der Bezeichnungen der Tabelle II skizziert, dann wurden in einer Tabelle den $C(i)$ und $FG(i)$ die zugehörigen Indizes des skizzierten Addierers zugeordnet, die Zahl dieser Indizes gezählt und endlich mit den Werten der Tabelle II der Elektronikaufwand berechnet. Dieses Verfahren läßt sich für $n = 6, 7, 8$ wiederholen, und sei zunächst auf die Addierer mit 4 τ Rechenzeit beschränkt. Für jeden Addierer sind 2 Versionen denkbar. Einmal der systematische nach Abb. 5, bei dem höchstens geringe Änderungen an der Spitze oder bei den notwendigen, aber nicht vollen Hilfsfunktionen stattfanden, und dann die geraffte Version, bei der alle Möglichkeiten zur Elektronik-Einsparung wahrgenommen wurden. Beide Zahlentripel sind für $n=5$ bis 8 errechnet und, erweitert um die Summe der beiden ersten Zahlen, nach abnehmendem Elektronikaufwand angeordnet.

77,	215,	954	4τ	292	n=8	systematisch
78,	209,	913	4τ	287	n=7	"
79,	204,	856	4τ	283	n=6	"
81,	199,	807	4τ	280	n=5	"
81,	195,	787	4τ	276	n=5	gerafft
80,	192,	780	4τ	272	n=6	"
79,	190,	778	4τ	269	n=7	"
80,	188,	776	4τ	268	n=8	"

Diesen Zahlen kann entnommen werden, daß der systematisch aufgebaute Addierer je mehr Elektronik verbraucht, je größer n wird. Besonders bei den Dioden ist die Steigerung stark. Da mit Elektronik mit hohem FAN-IN auch Schaltungen für niedrigeres FAN-IN aufgebaut werden können, wäre es sinnvoll, wollte man einen systematischen Addierer aufbauen, diesen für n=5 aufzubauen, selbst wenn das FAN-IN der Elektronik mehr zulassen würde. Legt man aber Wert darauf, wenig Elektronik zu benutzen, so läßt sich dies am besten mit möglichst großem n erreichen, weil sich bei großem n alle Einsparungs-Verfahren viel kräftiger auswirken. Auf der anderen Seite ist zu sehen, daß die Aufwandzahlen für den gerafften Fall weniger stark von n abhängig sind, es ist nur die Tendenz zu sehen, es findet keine wesentliche Einsparung durch Änderung von n statt.

Etwa die gleichen Ergebnisse werden erhalten, wenn für 5τ Rechenzeit und für n = 2, 3, 4, 5 die entsprechenden Zahlen berechnet werden. Man erhält:

84,	206,	820	5τ	290	n=4	systematisch
85,	185,	696	5τ	270	n=3	"
93,	190,	686	5τ	283	n=2	"
92,	188,	680	5τ	280	n=2	gerafft
81,	169,	636	5τ	250	n=3	"
79,	165,	633	5τ	244	n=4	"
79,	165,	632	5τ	244	n=5	"

Eine Ausnahme machen die Zahlen für $n=2$. Sie sind größer als die für $n=3$. Nimmt man sie aus dieser Aufstellung heraus, dann nimmt der Elektronikaufwand von oben nach unten ab.

Die Ausnahme für $n=2$ kann durch die Tatsache erklärt werden, daß in diesen Fällen sowohl das linke Ende der 2. Stufe wie auch fast das ganze linke Ende der 3. Stufe benötigt wird. Diese Enden benötigen, wie erwähnt, besonders viel Elektronik.

Für $n=5$ läßt sich nur ein geraffter Addierer bilden, sein Elektronikaufwand ist nur um eine Diode besser als der für $n=4$ (gerafft). Versucht man für $n=6$ ebenfalls einen 5τ -Addierer zu bauen, dann nimmt der Elektronikaufwand wieder zu. Betrachtet man Anlage 2, dann sieht man, daß dort die maximale FAN-IN-Zahl 6 nur für C_5 und für F, G_6^{11} benötigt wird. Geht man von dieser Anlage 2 aus und versucht, eine Elektronik sparende Änderung, wobei ein FAN-IN von 7 zugelassen sein soll, durchzuführen, dann ergibt sich kaum eine Möglichkeit dazu.

War es bei den 4τ -Addierern günstiger, n möglichst groß zu wählen, so zeigt es sich hier, wenn ein theoretisch möglicher 4τ -Addierer (für $n=5$ und 6) in einen 5τ -Addierer umgewandelt wird, daß die Forderung nach einem großen n nicht mehr erhoben werden muß.

Berücksichtigung von FAN-OUT-Zahlen

Als Abschluß soll wenigstens in einem Fall eine Lösung angegeben werden, bei dem das FAN-OUT-Problem berücksichtigt ist. Die Elektronik soll aus Invertern bestehen, siehe Abb. 8e, deren maximales FAN-IN und maximales FAN-OUT mit 8 angegeben ist. Am Eingang der Inverter kann Dioden-Logik OR-, AND- oder kombinierte Glieder bilden. Bei den kombinierten Gliedern dürfen jedem der 8 Eingänge eines OR-Gliedes noch 8 AND-Glieder vorgeschaltet sein. Durch den Inverter wird die invertierte logische Funktion erzeugt, es muß entweder ein zweiter Inverter folgen oder es muß die invertierte logische Funktion an den Eingang gegeben werden. Bei der Bildung der G_1 - und F_1 -Funktionen ist das letzte naheliegend. Wenn außerdem noch sichergestellt ist, daß das maximale FAN-OUT von 8 nicht überschritten wird, kann so verfahren werden, falls der gebildete logische Wert aber an mehr als 8 Stellen benötigt wird, können einem zweiten Inverter bis zu 7 Inverter parallel geschaltet werden, so daß der Funktionswert 64 mal zur Verfügung stehen würde. Die Zeiten für den Durchgang der Information durch OR- und AND-Glieder sind nicht mehr unterscheidbar, aber die Definition für τ ist, wie Abb. 8e zeigt, sinnvoll und meint die Zeit, in der 2 Sätze von Dioden und 2 Inverter durchlaufen werden.

Der betrachtete Addierer soll wieder 25 Bit haben, n ist nach der obigen FAN-IN-Voraussetzung gleich 7. Die Teile dieses Addierers, der in 2 Stufen arbeitet, sind schon vorher angegeben worden. Die erste Stufe ist in (29) angegeben und muß nur durch C_7 vervollständigt werden. Die Fortsetzung findet man in der aus (30) abgeleiteten Anordnung (31). Fügt man dann das erste volle Funktionspaar F_8^{15} und G_8^{15} hinzu, so folgt der Rest des Addierers in (24) und die Spitze in (27). Schreibt man alle 25 logischen Gleichungen für die C_1 und dazu die für die 4 Hilfsfunktionenpaare hin, so könnte man das Auftreten jedes F_1 und jedes G_1 abzählen und daraus die FAN-OUT-Zahlen für die F_1 und G_1 erhalten. Es werden genau 6 F_1 mehr als 8 mal gebraucht: 9 x F_{10} , 11 x F_{12} , 12 x F_{13} , 11 x F_{14} , 10 x F_{20} und 10 x F_{21} . Für alle G_1 ist die FAN-OUT-Zahl hinreichend. Der ungünstigste Fall liegt bei F_{14} vor. 4 x wird F_{14} in der logischen Funktion für C_{14} benötigt, und 7 x bei der Bildung der Paare F_8^{15} und G_8^{15} . Diese letzteren werden um die Zeit τ früher be-

nötigt als die für die Bildung von C_{14} . Hängt man an 7 Ausgänge von F_{14} diese 7 Eingänge für die Hilfsfunktionen und an den achten Ausgang ein in Reihe geschaltetes Inverterpaar, so erhält man am Ausgang dieses Inverterpaares die notwendigen F_{14} zum richtigen Zeitpunkt für die Bildung von C_{14} . Bei den anderen aufgeführten F_i wird ebenso verfahren, die für die Hilfsfunktionen benötigten FAN-OUT sind alle kleiner als 7, so daß immer ein Inverterpaar angeschlossen werden kann.

Mit diesen 6 Inverterpaaren ist das FAN-OUT aller F_i und G_i kleiner gleich 8, es braucht keine Vervielfachung innerhalb der Ebene der F_i und G_i stattzufinden. Es empfiehlt sich daher, alle F_i und G_i mit nur je einem Inverter zu bilden, d.h. die logische Funktion dieser Werte wird invertiert. Es werden gebildet $\bar{F}_i = \bar{A}_i \cdot \bar{D}_i$ und $\bar{G}_i = \bar{A}_i + \bar{D}_i$, woraus nach der Invertierung F_i und G_i wird. Die Zeit zur Erzeugung dieser Werte ist also kleiner als τ .

Die nächsten FAN-OUT, die betrachtet werden müssen, sind die von C_7 , dem letzten Carry der 1. Stufe, und die FAN-OUT aller Hilfsfunktionen F_8^r, G_8^r . C_7 geht in die Carry-Bildung von 18 weiteren Carry ein, dazu kommt noch die Verwendung von C_7 bei der Bildung von S_8 . C_7 wird aber aus einer Diodenfunktion und 2 nachgeschalteten Invertern gebildet, das gilt auch für alle Hilfsfunktionen. Es können also an den ersten dieser Inverter, der $\bar{C}_7$ am Ausgang hat, 3 weitere Inverter angeschlossen werden, so daß 24 weitere Werte C_7 bereitstehen. Genauso verfährt man bei $F_{20}^{23}, F_{16}^{19}, F_8^{15}$ und G_8^{15} mit je einem weiteren Inverter. Zur Erfüllung der maximalen FAN-OUT-Zahl waren also insgesamt 19 Inverter über die hinaus notwendig, die sowieso zum Aufbau des Addierers benötigt werden.

Die Berechnung des Aufwands für die Carrybildung nach dem angegebenen Verfahren liefert

29, 90, 429.

Diese Zahlen müssen jetzt anders interpretiert werden. Den 29 OR-Gliedern entsprechen, da alle Inverter dabei nur paarweise gebraucht werden, 58 Inverter. In den 90 AND-Gliedern sind 4 Hilfsfunktionen enthalten, die nicht einem OR-Glied vorgeschaltet sind, sie brauchen weitere 8 Inverter. Zu den insgesamt 66 Invertern kommen dann noch

die 19 für die Einhaltung der maximalen FAN-OUT-Zahlen. Das scheint viel zu sein, kann aber auch daran gemessen werden, daß allein 50 Inverter für die F_1 und G_1 und mindestens 25 Inverter für die S_1 benötigt werden, wenn sie gemäß (1) gebildet werden.

Der Gesamtablauf der Signale läuft durch maximal 6 Inverter. Z.B. 1 Inverter zur Erzeugung von F_1 , 2 weitere Inverter zur Erzeugung von C_7 , 2 weitere zur Erzeugung von C_8 und einen weiteren zur Erzeugung von S_9 . Der ebenfalls notwendige Wert von $\bar{C}_8$ bei der Bildung von S_9 könnte nach dem ersten Inverter bei der Bildung von C_8 entnommen werden. Nach der angegebenen Definition beträgt die Gesamtadditionszeit etwa mehr als 3τ , theoretisch wäre 4τ richtig. Praktisch kommen auch 4τ heraus, wenn in diesem Verlauf noch irgendwo zwei Inverter ohne Dioden eingeschoben werden. In der Regel wird mit S_1 auch $\bar{S}_1$ bestimmt, dazu werden alle C_1 und $\bar{C}_1$ an mehreren Stellen für den Gebrauch bei der Multiplikation und Division gebildet, so daß zur Vervielfachung der C_1 und $\bar{C}_1$ leicht echte 4τ erreicht werden.

Bei den anfangs genannten F_1 , deren FAN-OUT 8 überstieg, war kein F_1 dabei, das zur 1. Stufe nach (29) beitrug. Die Werte für das FAN-OUT der F_1 von F_7 bis F_1 sind 7, 8, 7, 8, 7, 6, 4.

Würde statt (29) die Anordnung (28) benutzt werden, so wären die Werte der benötigten FAN-OUT in der gleichen Reihenfolge 7, 12, 15, 16, 15, 12, 7.

5 FAN-OUT wären zu klein und die vorher angegebene Methode wäre nicht brauchbar, wenn die Carry der 1. Stufe gleichzeitig gebildet werden sollen.

So wie bei diesem Beispiel haben auch an anderen Teilen dieses und anderer Addierer die angewandten Methoden zur Elektronikeinsparung gleichzeitig die Anforderungen an die FAN-OUT-Zahlen heruntergesetzt. Die Frage nach einer Konstruktionsvorschrift für Addierer unter gleichzeitiger Berücksichtigung von FAN-IN- und FAN-OUT-Zahlen bleibt zunächst offen.

Zusammenfassung

In der vorliegenden Arbeit wird eine Relation hergeleitet, die für eine vorgegebene maximale FAN-IN-Zahl die drei Begriffe

Bit-Zahl eines Addierers
maximale Additionsgeschwindigkeit
Zahl und Typ von Hilfsfunktionen

verknüpft (Abb. 5).

Aus diesen Überlegungen folgt ein Konstruktionsprinzip für Look-ahead-Addierer durch sogenannte "volle" Hilfsfunktionen, die in der Regel noch durch "nicht volle" Hilfsfunktionen ergänzt werden müssen. Der so konstruierte Addierer besitzt gegenüber in der Literatur angegebenen Typen entweder höhere Geschwindigkeit oder geringeren Materialaufwand. Die "nicht vollen" Hilfsfunktionen können in gewissen Grenzen frei gewählt werden und regen dadurch an, die Auswahl so vorzunehmen, daß der Elektronikaufwand reduziert wird.

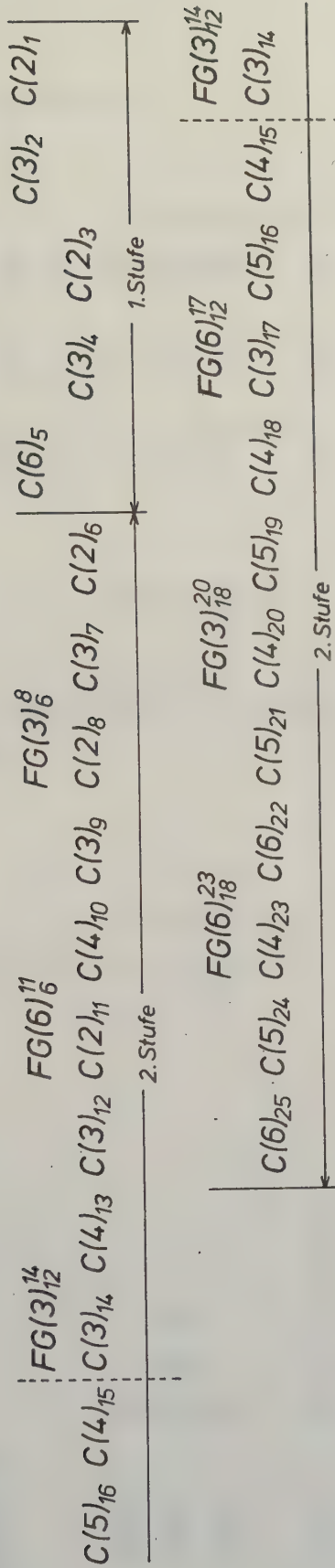
Ein dazu entwickeltes Minimisierungsverfahren wird anschließend für alle Bereiche des Addierers verwendet, um von einem "systematisch" aufgebauten Addierer zu einer "gerafft" aufgebauten Konstruktion überzugehen. Als überraschendes Ergebnis folgt, daß der relativ einfachste Look-ahead-Addierer, der aus mehreren nacheinander arbeitenden Stufen (Abb. 2) besteht, nicht nur langsamer ist, als ein durch Gebrauch von Hilfsfunktionen konstruierter Addierer, sondern daß dieser einfachere und langsamere Stufenaddierer einen höheren Elektronikaufwand benötigt. In Extremfällen kann dies bei großen FAN-IN-Zahlen bis zu 20% betragen.

An einem 25-Bit-Addierer wurde der Einfluß verschiedener maximaler FAN-IN-Zahlen auf den Elektronikaufwand untersucht. Beim systematisch aufgebauten Addierer steigt der Aufwand mit wachsendem maximalen FAN-IN, nach Anwendung aller Minimierungsmöglichkeiten haben in der "gerafften" Version die Addierer mit größerem FAN-IN den kleineren Aufwand, allerdings liegen alle Werte der gerafften Versionen

so nahe beieinander, daß der Nutzen von größeren FAN-IN als 6 für dieses Beispiel in Frage gestellt zu sein scheint.

Zu dem gleichen Schluß gelangt man, wenn der 25-Bit-Addierer mit so kleinem FAN-IN, daß eine Stufe mehr erforderlich ist, untersucht wird. Es wird, da hier mehr Möglichkeiten zur Minimierung vorhanden sind, generell mit etwas geringerem Aufwand gearbeitet werden können und es zeigt sich, läßt man für diese Stufenzahl größeren FAN-IN wieder zu, daß keine größeren FAN-IN als 6 praktisch benötigt werden. Natürlich sind alle diese Aussagen nur sicher für einen 25-Bit-Addierer.

Im letzten Abschnitt wird der Einfluß der FAN-OUT-Zahlen untersucht. An dem Beispiel des 25-Bit-Addierers läßt sich zeigen, daß hohe Anforderungen an FAN-OUT-Zahlen, die durchaus auftreten, deshalb keine unlösbaren Schwierigkeiten bieten, weil die Anschlußstellen die Werte nicht gleichzeitig oder nicht sofort benötigen, so daß zur Verringerung der FAN-OUT-Zahlen Vervielfacher-Verstärker eingebaut werden können. Der Addierer erforderte zur Einhaltung eines maximalen FAN-OUT von 8 zusätzlich 19 weitere Inverter, d.h. circa 10% Mehraufwand.

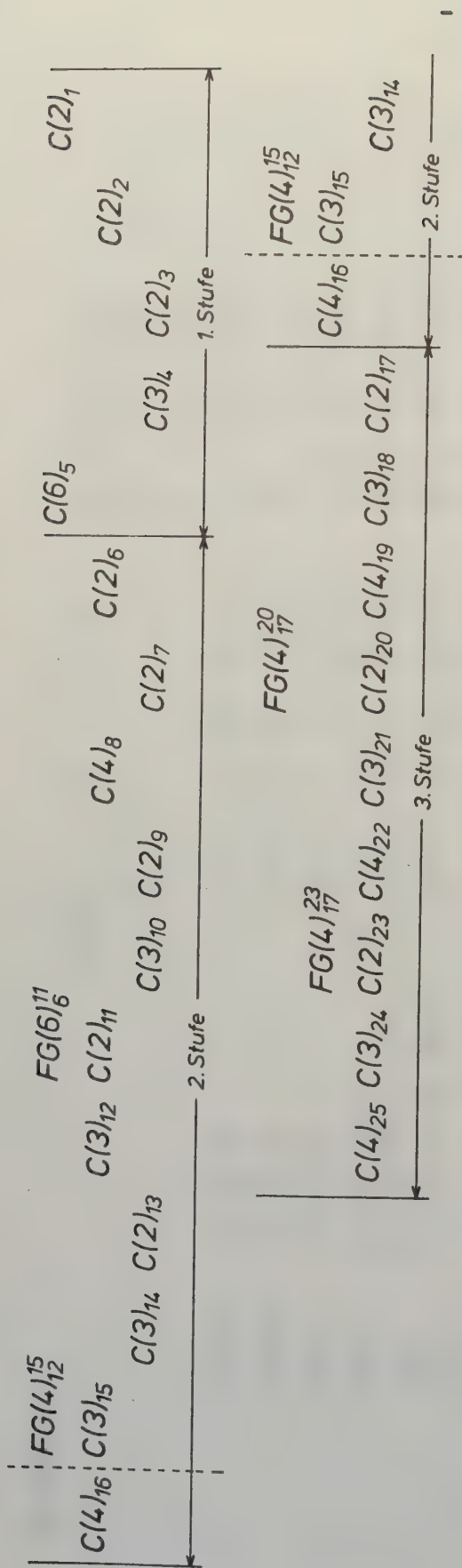


Indizes	Zahl der Indizes	Tabelle II
C(2) : 1, 3, 6, 8, 11	: 5 x (1, 1, 4) =	5, 5, 20
C(3) : 2, 4, 7, 9, 12, 14, 17	: 7 x (1, 2, 8) =	7, 14, 56
C(4) : 10, 13, 15, 18, 20, 23	: 6 x (1, 3, 13) =	6, 18, 78
C(5) : 16, 19, 21, 24	: 4 x (1, 4, 19) =	4, 16, 76
C(6) : 5, 22, 25	: 3 x (1, 5, 26) =	3, 15, 78
FG(3) : $\begin{smallmatrix} 8 & 14 & 20 \\ 6 & 12 & 18 \end{smallmatrix}$	: 3 x (1, 3, 11) =	3, 9, 33
FG(6) : $\begin{smallmatrix} 11 & 17 & 23 \\ 6 & 12 & 18 \end{smallmatrix}$	: 3 x (1, 6, 32) =	3, 18, 96
		31 95 437
		50 100 350
		81 195 787

(5''')

Anlage 1

25 Bit 4τ n=5



80

Zahl der
Indizes

Tabelle II

$C(2) : 1, 2, 3, 6, 7, 9, 11, 13, 17, 20, 23 : 11 \times (1, 1, 1, 4) = 11, 11, 44$
 $C(3) : 4, 10, 12, 14, 15, 18, 21, 24 : 8 \times (1, 2, 8) = 8, 16, 64$
 $C(4) : 8, 14, 19, 22, 25 : 5 \times (1, 3, 13) = 5, 15, 65$

$C(6) : 5 : 1 \times (1, 5, 26) = 1, 5, 26$
 $FG(4) : 15, 20, 23 : 3 \times (1, 4, 17) = 3, 12, 51$
 $FG(6) : 11, 6 : 1 \times (1, 6, 32) = 1, 6, 32$

$29, 65, 282$
 $50, 100, 350$
 $79, 165, 632$

$(5''')$
 (43)

25 Bit 5τ n=5

Anlage 2

Literatur

- [1] A. Weinberger, A One Microsecond Adder Using
 J.L. Smith, One-Megacycle Circuitry.
 IRE trans. electr. computers,
 ec. 5,2. S. 63 - 73, 1956

- [2] O.L. MacSorley High-Specel Arithmetic in Binary
 Computers.
 Proc. IRE, S. 67 - 91, 1961

- [3] Ivan Flores The Logic of Computer Arithmetic.
 Prentice-Hall, 1963

- [4] Hans Jörg Tafel Einführung in die digitale Daten-
 verarbeitung.
 Carl Hanser Verlag, München, 1971

- [5] L. Bauer Moderne Rechenanlagen.
 J. Heimhold Teubner, Stuttgart, 1965
 K. Samelson
 R. Sauer

Lebenslauf

Am 29. 2. 1924 in Berlin geboren, besuchte ich das "Berlinische Gymnasium zum Grauen Kloster" von 1934 bis 1942, Abschluß mit Reifevermerk wegen Einberufung zum Wehrdienst. Ab 1949 studierte ich Physik und Mathematik an der Universität Frankfurt am Main. Abschluß mit dem Diplom für Physik im Jahre 1957. Diplomarbeit auf dem Gebiet der Regelungstechnik.

Nach kurzer Industrietätigkeit Mitarbeit am Institut für Steuer- und Regeltechnik der Deutschen Versuchsanstalt für Luftfahrt in Oberpfaffenhofen bei München. Beschäftigung mit Compiler-Problemen.

Ab 1963 am Lehrstuhl für Regelungstechnik der Technischen Universität Berlin als Oberingenieur tätig, ab 1965 als Lehrbeauftragter für die Ausbildung im Gebrauch von ALGOL.

Anfertigung eines ALGOL-Compilers.

Ab 1969 am Lehrstuhl für Elektrische Steuerung und Regelung der Ruhr-Universität Bochum als Oberingenieur tätig.

some
it
these
also
the

very
in
the

